Fall 12-14-2010

Dynamic Data Driven Application System for Wildfire Spread Simulation

Feng Gu
Georgia State University

DYNAMIC DATA DRIVEN APPLICATION SYSTEM FOR WILDFIRE SPREAD SIMULATION

by

FENG GU

Under the Direction of Xiaolin Hu

ABSTRACT

Wildfires have significant impact on both ecosystems and human society. To effectively manage wildfires, simulation models are used to study and predict wildfire spread. The accuracy of wildfire spread simulations depends on many factors, including GIS data, fuel data, weather data, and high-fidelity wildfire behavior models. Unfortunately, due to the dynamic and complex nature of wildfire, it is impractical to obtain all these data with no error. Therefore, predictions from the simulation model will be different from what it is in a real wildfire. Without assimilating data from the real wildfire and dynamically adjusting the simulation, the difference between the simulation and the real wildfire is very likely to continuously grow. With the development of sensor technologies and the advance of computer infrastructure, dynamic data driven application systems (DDDAS) have become an active research area in recent years. In a DDDAS, data obtained from wireless sensors is fed into the simulation model to make predictions of the real system. This dynamic input is treated as the measurement to evaluate the output and adjust the states of the model, thus to improve simulation results. To improve the accuracy of wildfire spread simulations, we apply the concept of DDDAS to wildfire spread simulation by dynamically assimilating sensor data from real wildfires into the simulation model.

The assimilation system relates the system model and the observation data of the true state, and uses analysis approaches to obtain state estimations. We employ Sequential Monte Carlo (SMC) methods (also called particle filters) to carry out data assimilation in this work. Based on the structure of DDDAS, this dissertation presents the data assimilation system and data assimilation results in wildfire spread simulations. We carry out sensitivity analysis for different densities, frequencies, and qualities of sensor data, and quantify the effectiveness of SMC methods based on different measurement metrics. Furthermore, to improve simulation results, the image-morphing technique is introduced into the DDDAS for wildfire spread simulation.

INDEX WORDS: Wildfire spread, Modeling, Simulation, DEVS, DDDAS, Sequential Monte Carlo methods

DYNAMIC DATA DRIVEN APPLICATION SYSTEM FOR WILDFIRE SPREAD SIMULATION

by

FENG GU

A Dissertation Submitted in Partial Fulfillment of the Requirement for the Degree of

Doctor of Philosophy

In the College of Arts and Sciences

Georgia State University

2010

DYNAMIC DATA DRIVEN APPLICATION SYSTEM FOR WILDFIRE SPREAD SIMULATION

by

Feng Gu

Committee Chair: Xiaolin Hu

Committee: Rajshekhar Sunderraman

Saeid Belkasim

Jiawei Liu

Electronic Version Approved:

Office of Graduate Studies

College of Arts and Sciences

Georgia State University

December 2010

ACKNOWLEDGEMENTS

First of all, I thank my advisor Dr. Xiaolin Hu for his supervision in the past four years. He is an excellent researcher, full of creativity and innovative. He taught me how to express my ideas. He guided me to approach a research problem and persistently accomplish the goal. I was deeply impressed by his enthusiasm for research. Besides my advisor, I would like to thank my dissertation committee, Dr. Rajshekhar Sunderramn, Dr. Saeid Belkasim, and Dr. Jiawei Liu, for their insightful comments. During the course of my dissertation, I was supported in part by grants CNS-0841170 and CNS-0941432 from the National Science Foundation.

Also, I thank my family, my father Jinxue Gu, my mother Xiumei Wen, for their endless love and support, my sisters Shufen Gu and Shufang Gu, for their believing in me. I never feel my mother has left, giving me bravery, courage, and strength. Last but not least, I express my thanks to Yinting Xu, for his encouraging and trusting me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

1.1 Problem statement

Wildfires have significant impact on both the ecosystems and human society. The effects on ecological systems include burning local plants, reducing species diversity due to the emission of carbon dioxide, destroying organic nutrients to cause flash floods, and leading to climate changes by releasing carbon into atmosphere (Keeley 1995; Lindsey 2008; Kennard 2008; Running 2008). Wildfires also cause massive losses of natural forest resources, endangered species, properties, and even human lives. It is estimated that more than 11,000 communities close to federal land are subject to threats from wildfires in the US (Rey 2004). In the 2007 wildfire season, over 85,500 fires across the whole US burned more than 9.3 million acres of land. It cost 1.8 billion dollars in effort to fight wildfires and a potential 2.5 billion dollars in insured loss in California alone (Grossi 2007). To effectively manage wildfires, simulation models are used to study and predict wildfire spread. Over the years, several major wildfire spread simulation models have been developed, including FARSITE (Finney 1998), BehavePlus (Andrews et al. 2005), DEVS-FIRE (Ntaimo et al. 2008), and HFire (Morais 2001).

The accuracy of wildfire spread simulations depends on many factors, including GIS data, fuel data, weather data, and high fidelity wildfire behavior models. Unfortunately, due to the dynamic and complex nature of wildfire, it is impractical to obtain all these data with no error. For example, the weather data used in the simulation is typically obtained from local weather stations in a time-based manner (e.g., every 10 minutes). Before the next data arrives, the weather is considered unchanged in the simulation model. This is different from the reality where the real weather constantly changes (e.g., due to the interactions between wildfires and the weather). The GIS data and fuel data also have errors and are constrained by their spatial

resolutions. Besides data errors, the wildfire behavior model introduces errors too because of its computational abstraction. Due to these errors, the predictions from the simulation model will be different from what it is in a real wildfire. Without assimilating data from the real wildfire and dynamically adjusting the simulation, the difference between the simulation and the real wildfire is very likely to continuously grow.

With the development of sensors technologies and the advance of computer infrastructure, dynamic data driven application systems (DDDAS) become an active research area in recent years. In a DDDAS system, the data obtained from wireless sensors is fed into the simulation model to make predictions of the real systems. This dynamic input is treated as the measurement to evaluate the output and adjust states of the model. Based on these measurements, we can evaluate, choose, or analyze the system states utilizing statistical tools, data processing, and numeric or non-numeric techniques to improve the simulation results. To improve the accuracy of wildfire spread simulations, we also introduce the concept of DDDAS, which dynamically assimilates sensor data from real wildfires into the model to produce a time sequence of assimilated states. The assimilation systems map data between the model and the observation data and evaluate the system states to obtain the state estimations.

1.2 Structure of DDDAS

In a typical DDDAS, there exist three major components, including the application (the model system), the measurement model, and data assimilation methods. Therefore, it generates a wealth of new challenges for applications, algorithms, performances, and measurement approaches. For the wildfire spread simulation, the DEVS-FIRE model is developed as an integrated environment for surface wildfire spread and containment, built on Discrete Event System Specification (DEVS) formalism (Zeigler et al. 2000). Based on this wildfire spread

model, the real time data on-site is considered to be introduced to improve the simulation results. Therefore, the measurement model, which is used to couple the application model and real time data, should be developed, thus to compare the application model's output with the real data, and further estimate the real state of the system.

To effectively utilize the real time data in the system model and the measurement model, data assimilation methods that assimilate sensor data from real wildfires are needed. Data assimilation is an analysis technique, in which the observed data is accumulated into the model to produce a time sequence of assimilated states (Bouttier and Courtier 1999). Given a state space model of a system, related approaches are designed to estimate a system state from the observation data, including the simple analysis method and the statistical approach. The simple analysis method simply and directly utilizes the observation data for the state estimation of the system without considering all kinds of errors. Instead, the statistical approach assumes the estimations from the system model and the observations have errors and all the error are unknown but known distributed. Using these statistical tools, we try to find the estimation of the state given the observation data.

As described above, the structure of DDDAS for wildfire spread simulation is displayed in Figure 1.1. From Figure 1.1, we can see that the key components of the dynamic data driven system for wildfire spread model are the system model, the measurement model, and the data assimilation system. Basically, we use the physical model to predict the fire spread if there are the input sets including fuel data, GIS data, and weather data available. This static wildfire spread model is the basis of fire prediction. The measurement model is the connection between the wildfire spread model and the real time data. Fire sensors are deployed in the fire space and the corresponding data is collected. Feeding these data into the measurement model, we output

the data to the data assimilation system, which is compared with the real data from sensor networks. Consequently, the data assimilation system chooses the fire states as the inputs of DEVS-FIRE, continuing to execute the system model to predict the next fire state. Because we consider the uncertainty of fire states in the system, which will introduce a lot of computations, the whole system is constructed on the multicomputer environment, in which multiple computational units run the simulations in parallel to improve the performance.

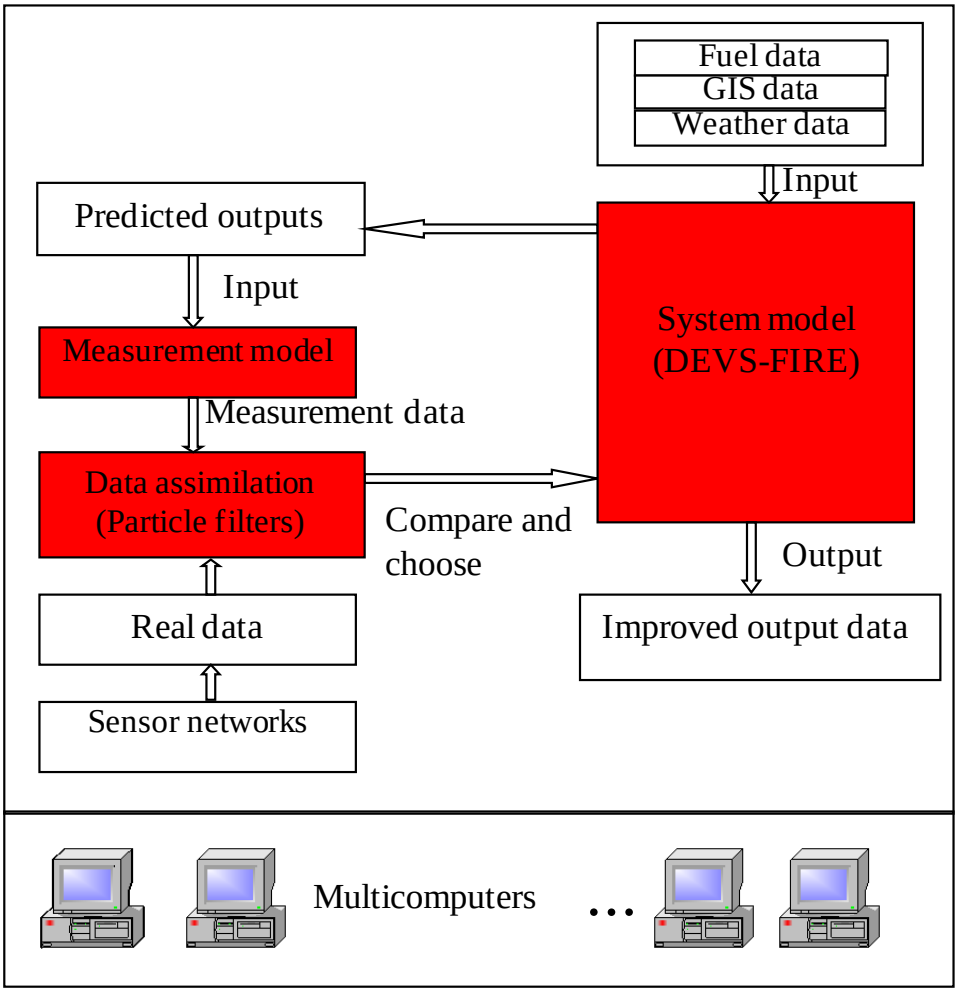


Figure 1.1 Structure of the dynamic data driven system for wildfire spread simulation

1.3 The Organization of the work

Based on the structure of DDDAS, the work will construct the entire system consisting of all the components, which will be explained later. Chapter 2 introduces the related work of data assimilation, wildfire spread models, sequential Monte Carlo methods, and dynamic data driven application systems. Chapter 3 describes DEVS-FIRE, the wildfire spread model we use in this work. Chapter 4 explains the measurement model, which maps between the system model and real time data. In Chapter 5, the data assimilation system is presented. Using identical twin methods, we design corresponding experiments to test the DDDAS for wildfire spread simulation in Chapter 6. Also, sensitivity analysis is done based on the experimental results in Chapter 7, including errors between the real system and the simulated system, the density of deployed sensors, the frequency of sensor data, and the quality of sensor data. To evaluate the data assimilation method, related measurement metrics are introduced in Chapter 8, thus to provide guidelines to develop more advanced algorithms. Additionally, image morphing is borrowed to DDDAS to improve the simulation results in Chapter 9. Chapter 10 provides the validation work of DEVS-FIRE. Finally, Chapter 11 gives conclusions and future work.

CHAPTER 2 RELATED WORK

2.1 Applications and algorithms of data assimilation

2.1.1 Applications of data assimilation

Data assimilation is used in many different fields, such as geosciences, weather forecasting, hydrology, and other environmental systems. The purpose of data assimilation is to use observation information to improve state estimation of a system under study. It tries to find the solutions by minimizing the errors between the real system and the models. The optimal interoperation analysis techniques are widely used in data assimilation, such as three-dimensional variational analysis (3D-VAR) and four-dimensional variational assimilation (4D-VAR). Although both of them minimize the cost function to obtain optimal estimations, 4D-VAR incorporates a prediction model to compare the model state and the observations at different time steps. The work of Bei et al. (2008) proposed a data assimilation system to improve ozone simulations in Mexico City basin using 3D-VAR that generated the optimal estimate of the true atmospheric state during the analysis time. Wilkin et al. (2008) used satellite remotely sensed observations to the regional ocean modeling system in ocean analysis and prediction, in which 4D-VAR was the primary analysis technique. It combined observations and modeling to produce an optimal estimation of the real ocean state. Kalman filter (Kailath et al. 2000) is an analysis technique that estimates the state of a dynamic system with observations represented by a linear state space model. For applications with non-linear behaviors, the classical Kalman filter needs to be extended. Antoniou et al. (2007) studied three extensions of Kalman filter including extended Kalman filter, limited extended Kalman filter, and unscented Kalman filter, to find the solutions to non-linear discrete-time state space models.

In the wildfire spread simulation, less research exists in data assimilation. In the limited work that we are aware of, Bradley (2007) proposed an approach to estimate forest fires based on sequential Monte Carlo methods from video images. In this work, a blurring function was used to add uncertainty to the images, and the information from miniature air vehicles was used as the measurement data to estimate fire poses. No wildfire spread simulation model was used in this work. Another group of work (Mandel et al. 2004; Mandel et al. 2009; Douglas et al. 2006; Coen et al. 2007) used data assimilation to investigate fire behaviors of wildfire spread models. Their work was built on a reaction-diffusion-convection partial differential equation model and used ensemble Kalman filter as the estimation method. In their work, the real time data was assimilated into the fire model to estimate the temperature of each cell. Our previous work (Gu and Hu 2008; Gu et al. 2009; Yan et al. 2009) explored applications of sequential Monte Carlo methods to state estimation in wildfire spread simulation using a discrete event simulation model. Preliminary results showed that sequential Monte Carlo methods were promising techniques for supporting data assimilation in wildfire spread simulations.

2.1.2 Algorithms of data assimilation

The methods of data assimilation include the simple analysis approach and the statistical approach. The simple analysis method considers the observation as the “truth”, and directly and simply uses the observation data. The state of the model is set to the observation values close to available observations and to an arbitrary state otherwise. Cressman analysis is one of simple analysis methods. In Cressman analysis, the analysis is generated by interpolating between the background (the previous estimate of the model state) and the observations, in the vicinity of each observation value. The simple analysis method is the basic tool because of its simplicity. However, it has some disadvantages as discussed in (Bouttier and Courtier 1999). To solve these

problems (the existing errors of background and observations) and obtain quality estimations, better methods to combine uncertainty are needed.

In the statistical approach, we try to use all the information, but don't completely trust them. We can find a strategy to minimize the average of the differences between the analysis and the truth. In this sense, the analysis can be seen as the optimization problem. All the related errors are assumed to be unknown and have known statistical properties. There are two main ways to define the statistical analysis problem. The first one is to assume that the background covariance and error covariance are known, and derive the analysis equations according to the constraint that total analysis error variances are minimum. In the second approach, we hold the assumption that the probability density functions of the background and observation errors are Gaussian, thus to derive the analysis equations by obtaining the state with maximum probability. Both of the approaches lead to kinds of algorithms, directly determining the analysis gain matrix and/or minimizing a quadric cost function, although their numerical properties greatly differ.

The optimal interpolation (OI) is a technique to simply and directly compute the gain matrix using matrix operations. The basic assumption of OI is that for each model variable, only a few observations are important. This will raise the question how to decide the used observations for the model variables. Two common selection schemas are used including pointwise selection and box selection. In pointwise selection, each analysis point is only sensitive to observations within a small range and two neighboring points have different observation sets. For box selection, all points in an analysis box are located in a larger selection box and two neighboring analysis boxes have almost the same observations. The advantages of OI include easy implementation and small cost (if right selections are made). But it may produce errors because difference sets of

observations are used in different parts of the model state. For the small and large scale analysis, it possibly leads to inconsistency.

Instead of computing gain matrix, three-dimensional variational analysis (3D-VAR) tries to find an approximate solution to the equivalent minimization problem defined by the cost function, evaluating the cost function and its gradients several times. 3D-VAR is popularly used because of its simplicity and supporting complex observation operators. Also, it also accepts the external constraints. Four-dimensional analysis (4D-VAR) is a simple generalization of 3D-VAR, in which observations are distributed in time. In 4D-VAR, the observation operators include a forecast model used to compare the model state and the observations at the appropriate time steps. Comparing with 3D-VAR, 4D-VAR works only if the model is perfect.

Kalman filter and its extended version (extended Kalman filter) are developments of least squares analysis in the sequential data assimilation. The extended Kalman filter doesn't need the linear model operator and/or observation operator. Their inputs include the definitions of the model operator and the observation operator, the initial conditions, and the sequences of observations. Kalman filter is done in a recursive manner. The estimates from the previous step and the current observation are used for the current state estimation. Two basic steps are involved in Kalman filter including predict and update. The predict step uses the estimates from the last step to produce the current state. In the update stage, the current a prior prediction is combined by the observation, thus to refine the current estimate to produce a posterior prediction. Kalman filter has many common features with 4D-VAR. In essential, if the model is perfect with the same time interval and input data, 4D-VAR analysis at the last time step is equal to that of Kalman filter at the same time step. They also differ in many aspects: (1) 4D-VAR is computationally cheaper; (2) 4D-VAR is more optimal because it uses all the observation once

and is not sequential; (3) 4D-VAR runs for a finite time interval, whereas Kalman filter can implement forever if the observations at the next step are available; (4) 4D-VAR must meet the requirements that the model is perfect. Ensemble Kalman filter is from a version of Kalman filter with large number of variables. It is related to sequential Monte Carlo methods (particle filters) discussed in the next sections and adopted in this work. However, ensemble Kalman filter assume that all probability functions involved are Gaussian, and it is more efficient than sequential Monte Carlo methods. Note that the details of the most of algorithms above can be found in (Bouttier and Courtier 1999).

2.2 Wildfire spread modeling

Simulations of wildfire can be categorized into two approaches, the physical approach and the empirical approach. The physical approach considers fire spread as heat transfer between burning and unburned fuel using partial differential equations to solve for filtered fire spread (see (Douglas et al. 2006; Linn et al. 2002; Pastor et al. 2003; Weber 1991)). The empirical approach relies on statistical correlations between variables known to influence fire spread with field observations of rates of spread. A widely used empirical fire behavior model is Rothermel's model (Rothermel 1972). Several major wildfire simulation systems have been developed to date, including FARSITE (Finney 1998), BehavePlus (Andrews et al. 2005), and HFire (Morais 2001). These systems use Rothermel's fire behavior model to compute the rate of fire spread, and determine the fire size according to an elliptical shape. They are raster-based spatially explicit models, and use a discrete time approach for simulating the wildfire growth. Many complex simulation models allow the wildfire to feed back upon the atmosphere. FIRETEC, a wildfire behavior model developed at Los Alamos National Laboratory, explored the interactions between the fire model and wind conditions (Linn et al. 2002). In previous work, we developed a

discrete event wildfire simulation model called DEVS-FIRE (Ntaimo et al. 2008; Hu et al. 2010) that used Rothermel's fire behavior model too. The DEVS-FIRE model is described in more details in Chapter 3.

2.3 Sequential Monte Carlo methods and their applications

2.3.1 Overview of sequential Monte Carlo methods

Sequential Monte Carlo (SMC) methods are sample-based methods that use Bayesian inference and stochastic sampling techniques to recursively estimate the state of dynamic systems from some given observations. A dynamic system is formulated as discrete dynamic state-space model, which is composed of the system model of equation (2.1) and the measurement model of equation (2.2) (Jazwinski 1970) as shown below. In the equations, t is time step, s_t and m_t are the state variable and the measurement variable respectively, the functions of f and g define the evolutions of the state variable and the measurement variable, and v_t and w_t are two independent random variables to generate the state noise and the measurement noise.

$$s_{t+1} = f(s_t, t) + v_t. \tag{2.1}$$

$$m_t = g(s_t, t) + w_t. \tag{2.2}$$

For state estimation, one needs to seek estimates of s_t based on the set of all measurements $m_{1:t} = \{m_i, i = 1,2,...,t\}$. In Bayesian filtering, both the state and the measurement variables are stochastic variables. Assuming the probability density $p(s_{t-1} \mid m_{1:t-1})$ at time step t-1 is available, the prior probability density function of the state at time step t can be obtained using the system model as shown in equation (2.3), where $p(s_t \mid s_{t-1})$ is the system model. If the measurement at time step t is available, one can update the prior probability density function according to Bayes

theorem as shown in equation (2.4). In equation (2.4), $p(m_t \mid s_t)$ can be obtained by the measurement model, and $p(m_t \mid m_{1:t-1})$ is a normalizing constant that can be computed by equation (2.5) according to Bayes theorem and Markov property. Equation (2.3) and equation (2.4) form the foundation to recursively predict the prior probability density function, and update it to the needed posterior probability density function of the current state.

$$p(s_t \mid m_{1:t-1}) = \int p(s_t \mid s_{t-1}) p(s_{t-1} \mid m_{1:t-1}) ds_{t-1}. \tag{2.3}$$

$$p(s_t \mid m_{1:t}) = \frac{p(m_t \mid s_t) p(s_t \mid m_{1:t-1})}{p(m_t \mid m_{1:t-1})}. \tag{2.4}$$

$$p(m_t \mid m_{1:t-1}) = \int p(m_t \mid s_t) p(s_t \mid m_{1:t-1}) ds_t. \tag{2.5}$$

Because it is difficult to solve the multidimensional integrals, many approximation algorithms are proposed, among which SMC methods are described below.

SMC methods approximate the posterior probability density function $p(s_t \mid m_{1:t})$ by a set of samples and their corresponding weights. An important concept in SMC methods is the principle of sequential importance sampling (SIS). In SIS, the posterior probability function is approximated by equation (2.6), where $s_t^{(i)}$ and $wt_t^{(i)}$ are particle i at time step t and its normalized weight, and δ is the delta function. The weights are defined in equation (2.7), where $q(s_t^{(i)} \mid s_{t-1}^{(i)}, m_t)$ is the importance density, which can be easily generated.

$$p(s_t \mid m_{1:t}) \approx \sum_{i=1}^{N} wt_t^{(i)} \delta(s_t - s_t^{(i)}). \tag{2.6}$$

$$wt_t^{(i)} \propto wt_{t-1}^{(i)} \frac{p(m_t \mid s_t^{(i)}) p(s_t^{(i)} \mid s_{t-1}^{(i)})}{q(s_t^{(i)} \mid s_{t-1}^{i}, m_t)}. \tag{2.7}$$

However, sequential importance sampling has the limitation that the entire process relies on the initially generated samples. To improve the algorithm, a resampling step is added by using replicated particles in proportion to their weights for future use. This gives rise to the sequential importance sampling with resampling (SISR), which forms the basic structure of SMC methods. With SMC methods, it has been shown that a large number of particles are able to converge to the true posterior density even in non-Gaussian, non-linear dynamic systems (Crisan 2001). For systems with strongly non-linear behaviors, SMC methods are more effective than the widely used Kalman filter and its various extensions. More details about the algorithm can be found in Gordon et al. (1993).

To summarize, a basic SMC algorithm that implements the SISR procedure has multiple iterations. In each iteration, the algorithm receives a sample set $S_{t\text{-}1}$ representing the previous belief of the system state, and an observation m_t. In the importance sampling step, each sample in $S_{t\text{-}1}$ is used to predict the next state. This is done by sampling from the density $p(s_t|s_{t\text{-}1})$ that represents the system dynamics (which is the simulation model in the work). Then the importance weight of every sample is computed and normalized. Finally, in the resampling step, N offspring samples are drawn with probability proportional to the normalized sample weights. These samples represent the posterior belief of the system state and are used for the next iteration. To apply this algorithm to data assimilation for wildfire spread simulation, we need to formulate the problem accordingly and develop associated models and techniques following the algorithmic structure of SMC methods.

2.3.2 Applications of sequential Monte Carlo methods

SMC methods find applications in many problem domains, including signal processing, wireless communication, target tracking, speech recognition, computer vision, mobile robot

localization, and DNA sequence analysis (Chen 2004). Gustafsson et al. (2002) developed a framework and several algorithms for the problems of positioning, navigation, and tracking using SMC methods. Fox et al. (2001) used SMC methods to solve robot localization, an important problem for mobile robots. In Mihaylova et al. (2007), the authors proposed Monte Carlo techniques for mobility tracking in wireless communication networks in terms of the signal strength, by which the mobile station's position and speed can be correctly estimated. In Azzabou et al. (2005), SMC methods were used in image processing to improve the image quality. Other applications of SMC methods include biology and chemistry. Zhang et al. (2003) provided an application of SMC methods in biology, in which populations of compact long chain polymers were created by the Monte Carlo methods to study the relationships between packing density and chain length. The work of Chen et al. (2008) set up a probabilistic framework for the dynamic data rectification, which provided a basis for process fault diagnosis.

2.4 Applications of dynamic data driven application systems

Dynamic data driven application systems are widely used in many fields, such as engineering, crisis management and environmental systems, medical, manufacturing, business, and finance (Darema 2007). This is because the new paradigm introduces dynamic observations from real systems into the system model. As said by Derema (2000), DDDAS was possibly becoming the revolutionary concepts in science, engineering, and management systems.

In the engineering, the mechanism of dynamic data driven simulation is largely used in design and control of systems. Farhat et al. (2006) intended to improve active health monitoring, failure prediction, aging assessment, informed crisis management, and decision support for complex and degrading structural engineering systems by using dynamic data driven methods.

Wang (2004) presented a set of data driven hardware and software techniques to explore the input space for performance/energy optimization.

Additionally, in crisis management and environmental systems, the concept of data driven simulation is utilized to incorporate the real time data into the physical models. Fujimoto et al. (2006) applied dynamic data driven application systems to monitor and manage surface transportation systems. In the work, the hierarchical DDDAS architecture was presented including vehicle, roadside, and traffic management center simulations. For environmental systems, such as weather, hurricanes, and fire propagation, it is very important for people to effectively predict their states, thus to forecast, control, or suppress them. Allen (2007) tried to couple the real time sensor information with the water circulation models to forecast the emergency event of hurricane and highlight the challenges for accurate estimations of these events.

In the medical area, the dynamic data driven simulation can be used for illness treatment. In the laser treatment of cancer, the higher intensity heat source may be used to ablate the affected tissue. In order to precisely control the treatment process, the heat transfer computational model was developed to employ the real time data to optimize the control of the treatment. This dynamic data driven application system was capable of estimating and guiding the computer controlled temperature in the biological domain with very good accuracies (Oden et al. 2006).

Other important application domains of dynamic data driven application system are manufacturing, business, and finance. Flexible manufacturing systems, such as mass customization, require handling the product variety, uncertainty in the product demands, and reconfiguration of manufacturing resources. These requirements are just the purposes of dynamic data driven application systems. When the requirements change, the model can adjust and

reconfigure corresponding resources according to the latest analysis. Example of this kind of model can be found in (Qiao et al. 2003). With the increasing complexity of supply chain in enterprises, we need to resort to computer simulation to optimize the business process. To overcome the limitations of the conventional model, which is useful to experts, people need to simplify the model for use. Therefore, Tannock et al. (2007) introduced the data driven concept to automatically construct the model by the input data from the company. This provided multiple scenarios for employees to make their decisions.

From the recent work above, we can see that most applications with plentiful real time data are the potential fields, in which the dynamic data driven simulation will be deployed, for example, stock prediction with a huge real time data. Although there are many challenging tasks, such as the non-linearity of the data, the large computation complexity, statistical tools to process and analyze data, the dynamic data driven simulation is still a very active research topic in modeling and simulation.

CHAPTER 3 WILDFIRE SPREAD MODEL OF DEVS-FIRE

3.1 Concepts of DEVS

3.1.1 Framework of M&S

The conceptual framework of DEVS is shown in Figure 3.1. Basically, the modeling and simulation concerns three basic objects including experimental frame, model, and simulator. From the figure, we know that the real world is the fundamental data source, and the experimental frame specifies the constraints to define the system under experiment and study. The model is a series of instructions and structures to represent the real system, and the simulator can be seen as a device to execute the model. Modeling relation connects the real world and the model, which specifies how well the model represents the real world. Simulation relation, linking the model and the simulator, denotes how the simulator executes the instructions and structures.

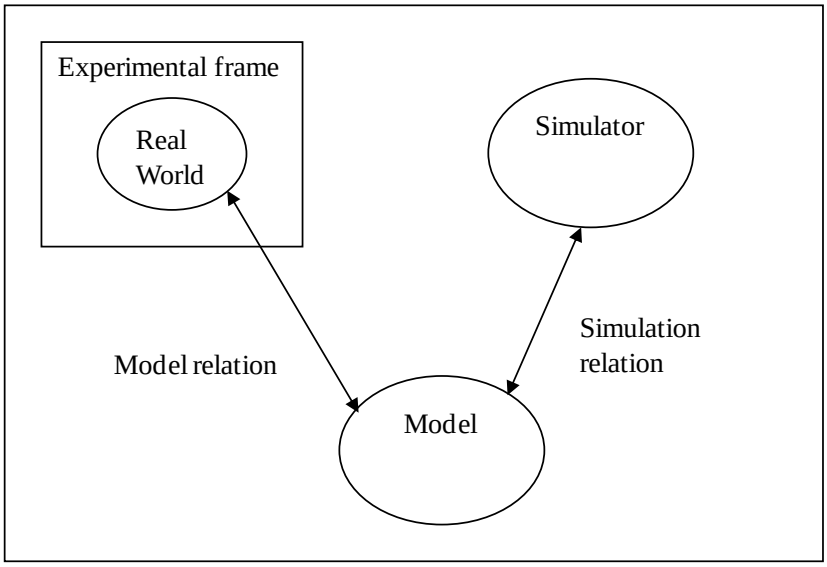


Figure 3.1 Framework of M&S

3.1.2 DEVS formalism

Formalism represents the structure of a model in mathematics. In DEVS formalism, mathematical models are used to define the events occurrence at different times. There are two kinds of models in DEVS formalism, the atomic model and the coupled model. An atomic model is a structure as shown in equation (3.1).

$$M =< X, S, Y, \delta_{\text{int}}, \delta_{ext}, \delta_{con}, \lambda, ta >, \tag{3.1}$$

where X is the set of input values; S represents a set of states; Y defines the set of output values; δ_{int}: $S \rightarrow S$ is the internal transition function; δ_{ext}: $Q \times X^b \rightarrow S$ is the external transition function, where $Q \in \{(s, e)| s \in S, 0 \leq e \leq ta(s)\}$ is the set of all the states, e is the elapsed time since the last transition, X^b denotes the collection of bags over X; δ_{con}: $S \times X^b \rightarrow S$ is the confluent transition function; λ: $S \rightarrow Y^b$ is the output function; ta: $S \rightarrow \mathbf{R}_0^+$ is the time advance function. A coupled model is composed of atomic models defined as shown in equation (3.2).

$$N =< X, Y, D, \{M_j\}, C_{ext}, C_{\text{int}}, C_{out}, Select >, \tag{3.2}$$

where X is the set of input events; Y is the set of output events; D is the name set of subcomponents; $\{M_j\}$ is the set of subcomponents, where for each $i \in D$, M_i is either an atomic model or a coupled model; $C_{ext} \subseteq X \times \bigcup_{i \in D} X_i$ is the set of external input couplings; $C_{int} \subseteq \bigcup_{i \in D} Y_i \times \bigcup_{i \in D} X_i$ is the set of input couplings; $C_{out} \subseteq \bigcup_{i \in D} Y_i \times Y^{\varphi}$ is the external output couplings function; $Select$: $2^D \rightarrow D$ is the tie-breaking function which defines how to select the event from the set of simultaneous events. The system can be modeled using DEVS formalism in a hierarchical way.

3.1.3 Applications of DEVS

Since it was proposed, DEVS has been used to model both continuous and discrete systems, such as knowledge-based control of industry products, autonomous agents systems, construction systems, supply-chain systems, systems biology, and hardware/software systems. In Le Goc and Frydman (2003), SACHEM, a real time intelligent diagnoses system based on DEVS paradigm was proposed. It provided a framework for knowledge-based control of steel production, and it was a successful case to use DEVS formalism in industry to save money up to millions of euros annually. In El-Osery et al. (2002), a virtual laboratory for multi-physics agents was built based on four components, computer networks, CORBA, DEVS, and soft computing methods (e.g., reasoning logic) from bottom to top. DEVS is also used in construction simulations, for example, Palaniappan et al. (2006) used DEVS framework to analyze the workflow between various trade contractors in production home building. To gain efficiency, it is necessary to study the dynamic changes in supply-chain systems. Therefore, DEVS is adopted in some supply-chain systems to analyze the complex interactions. Huang et al. (2009) presented an application of DEVS formalism in semiconductor manufacturing supply-chain systems, exploring the complex relationships between control policies and manufacturing processes. System biology focuses on analyzing the behavior and interrelationships between entities of entire functional biological systems. Towards this goal, Uhrmacher and Priami (2005) adopted both DEVS and π-calculus approaches to analyze multiple relationships to study the biological systems. In addition, hardware-in-the-loop applications can benefit from DEVS framework. In Glinsky and Wainer (2004), authors developed a technique to enable the transitions from the models to the actual

hardware, facilitate the testing purpose in a risky environment, and support component reuse in hybrid hardware/software systems using DEVS formalism.

Due to its wide use in many applications, DEVS has become a simulation tool in a variety of implementations. DEVSJAVA (Zeigler and Sarjoughian 2002) is an object oriented M&S environment based on Java language and DEVS formalism. Another implementation of DEVS is DEVSC++ (Zeigler et al. 1996). These provide conveniences for people to model the applications in their fields.

3.2 Wildfire spread model using DEVS — DEVS-FIRE

Based on DEVS formalism, the Rothermel's model is chosen in the wildfire spread simulation because it has been extensively validated and proven to be stable and robust (Rey 2004). The following subsections will explain some important aspects of DEVS-FIRE.

3.2.1 System architecture of DEVS-FIRE

The DEVS-FIRE model is an integrated environment for surface wildfire spread and containment, built on Discrete Event System Specification (DEVS) formalism (Zeigler et al. 2000). The overall structure of DEVS-FIRE is shown in Figure 3.2. From the figure we know that the fire spread model is the core of DEVS-FIRE, which is modeled as a cellar space containing neighbored cells with initialized fuels and the GIS data. When a cell is ignited, Rothermel's model (the Behave model) is used to compute the speed and direction of fire spread. Built on the fire spread model, DEVS-FIRE also supports fire containment simulation by connecting to the fire fighting model. More details about the fire containment simulation can be found in (Ntaimo et al. 2008). The visualization component displays the simulation results of DEVS-FIRE.

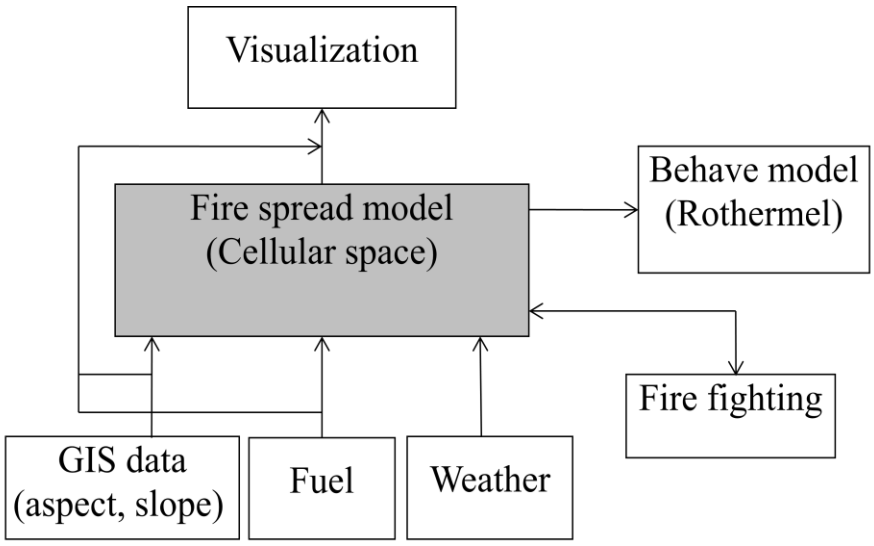


Figure 3.2 Structure of DEVS-FIRE

3.2.2 Cellular space model of DEVS-FIRE

In DEVS-FIRE, the forest is modeled as a two-dimensional cell space containing a series of rectangular cells whose dimensions depend on the resolution of the GIS fuel and the terrain data. The fuel, terrain, and weather conditions within individual forest cells are assumed to be constants (in the current model, the weather is assumed to be uniform for the entire area). Each cell can be seen as a DEVS atomic model during the duration of simulation and locally calculates the rate of speed and the direction based on the parameters of fuel model, slope, aspect, and the weather data. Each cell has eight neighboring cells according to the Moore neighborhood. The entire cell space is a coupled model by connecting input ports and output ports between neighboring cells, through which a cell can send messages to ignite its neighbor cells. The behaviors of burning cells depend on both external inputs from their neighbors and the dynamic

change of the weather data, which is globally passed on to all the cells in the cell space model.

Initially all the cells are set to the state of *unburned* (passive). If a cell receives an ignition message and its fireline intensity is larger than the burning threshold, its state will be changed to *burning*. After the burn delay time has elapsed, the burning cell changes to the *burned* state. This time delay depends on the size of the cell and the fire spread speed, which is computed using Rothermel's fire Behave model. As a result, fire spread is modeled as a propagation process that burning cells ignite their unburned neighboring cells. The state transitions of forest cells are displayed in Figure 3.3. In the figure, the state transitions for fire spread are explained. For the fire suppress part of DEVS-FIRE, related states (others) are needed, and not included here. Rothermel's fire behavior model is a semi-empirical mathematical model. In a small area and short-time periods, the fuel is taken to be homogeneous. The outputs of Rothermel's model include the rate of spread, the direction of maximum spread, flame length, and fire intensity that measures the rate of heat released by a fire. In DEVS-FIRE, the rate (and direction) of spread from Rothermel's model is then decomposed into eight directions including North, NorthEast, East, SouthEast, South, SouthWest, West, and NorthWest from the ignition point according to an elliptical shape as illustrated in Figure 3.4. The shape of this ellipse is computed by the midflame wind speed and the fire spread rate (see (Finney 1998)). Note that in Figure 3.4, we assume the maximum rate of spread is in the South direction. The fire spread model of DEVS-FIRE has been partially validated by comparing with results from FARSITE, which will be discussed in Chapter 10.

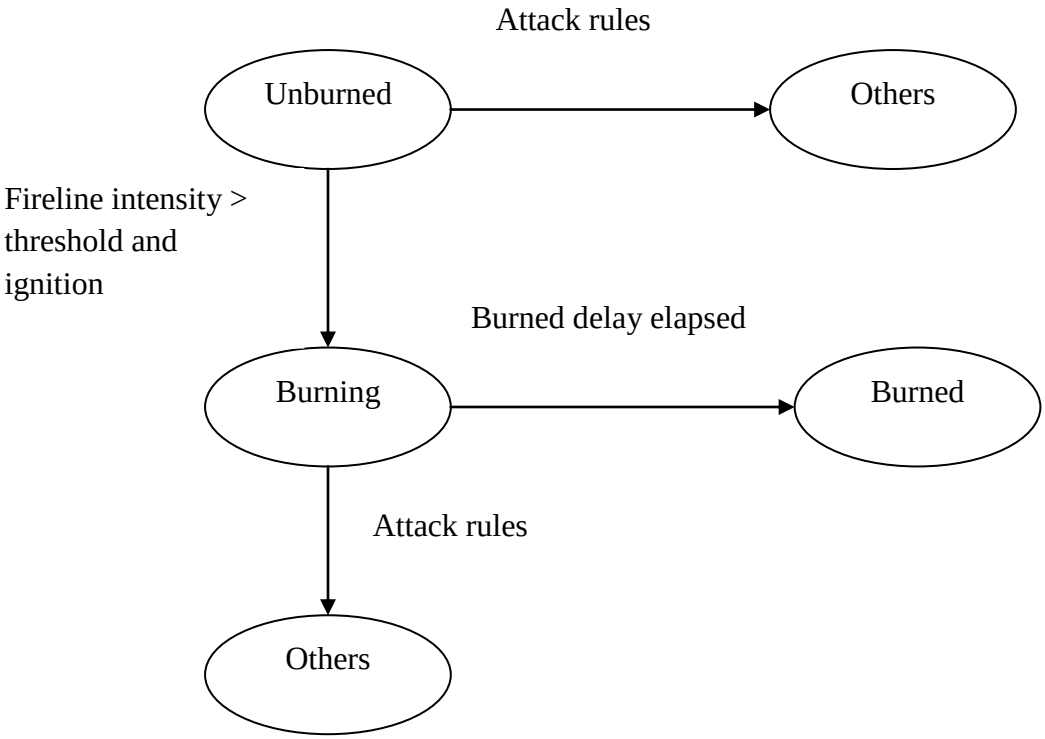


Figure 3.3 State transitions of forest cells

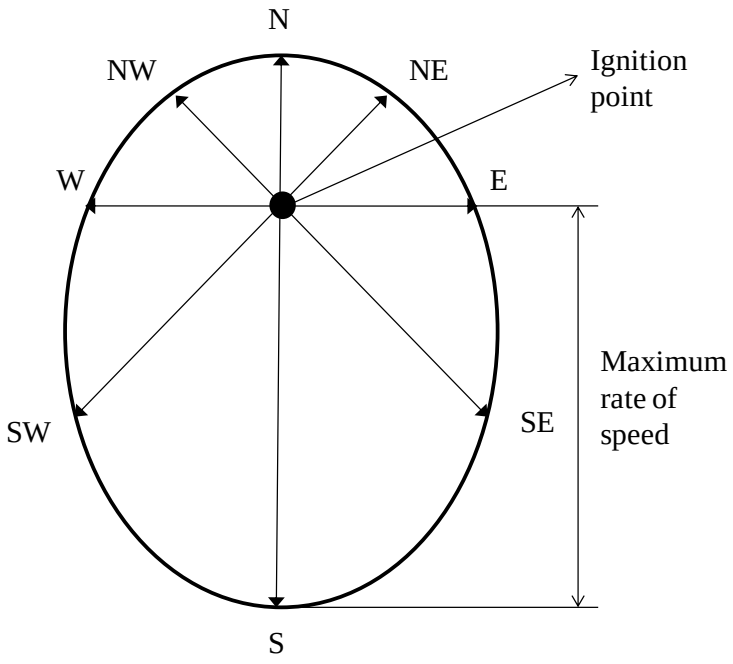


Figure 3.4 Decomposition schema of DEVS-FIRE

3.2.3 Specification of DEVS-FIRE

As described above, the fire spread model of DEVS-FIRE is a cellular space model. Each cell is modeled as an atomic model, and is coupled with its eight neighbors except the boundary cells to form a cell space. Therefore, the cell space is as a DEVS coupled model consisting of all the cells. In the DEVS formalism, a forest cell *FC* can be defined according to (Zeigler et al. 2000) as shown in equation (3.3).

$$FC = < X, Y, S, \delta_{ext}, \delta_{\text{int}}, \delta_{con}, \lambda, ta, xID, yID >, \tag{3.3}$$

where *X* is a set of inputs from its neighbors or weather inputs such as wind speed and direction used in calculation; *Y* defines the set of outputs, which contains the data fed into its neighbors for computation; *S* represents the cell's states, for example, *unburned*, *burning*, and *burned* during the fire spread; δ_{ext}, δ_{int}, and δ_{con} represent the external function, the internal function, and the confluent function respectively. These functions are used to specify the state transitions for the forest cell; λ refers to the output function, defining the outputs based on the cell's states; *ta* specifies the time advance; and *xID*, *yID* are the coordinates of the cell.

After coupling all the forest cells, we form a forest cell space coupled model, which can be specified as shown in equation (3.4) according to the DEVS formalism.

$$FCS = < X, Y, D, B, N, C_{ext}, C_{\text{int}}, C_{out}, r >, \tag{3.4}$$

where *X* and *Y* are the set of input events and output events respectively; *D* defines the set of all the indexes in the cell space; *B* refers to all the boundary cells; *N* specifies the neighborhood of the cell space, which is the Moore neighborhood, 8 neighbors for each cell in DEVS-FIRE; C_{ext}, C_{int}, and C_{con} are internal couplings, external input couplings, and external output couplings respectively; *r* is defined as the resolution of the cell space,

which is the size of each cell. The used DEVS-FIRE model was implemented in the DEVSJAVA environment (Zeigler and Sarjoughian 2002).

3.2.4 Interface of DEVS-FIRE

From the structure of DEVS-FIRE, we know that the inputs should be specified including the slope, the aspect, the fuel model, the weather condition, ignition points, and running time. In DEVS-FIRE, we use a cellular space to represent the fire field. Therefore, we also need to define the sizes of cellular space and each cell. Then, to define the slope, aspect, and the fuel model of this cell space, three text files are needed. In each file, a matrix whose dimensions are the number of cells horizontally and vertically is used to specify slope, aspect, and fuel model values of all forest cells in the cell space. The weather conditions are also defined by a text file, in which the wind speeds, wind directions, and the moistures at different times are contained. During the two adjacent time steps, the weather conditions are treated as the same. The ignition points are given by the coordinates of cells that are initially burned. The running time means the time period of the DEVS-FIRE's execution.

The outputs of DEVS-FIRE include fireline intensities, burning perimeters, burning areas, etc. of the cells. The graphical output of the fire spread is a fire map, which represents all the cells' states after the running time. In addition, the burning perimeters and burning areas of can be captured at intervals. For example, assume we have a cell space of 382 × 266 whose cell size is 30 × 30, and the aspect, the slope, and the fuel model are defined using three text files, which are displayed in Figure 3.5. If the weather conditions are that the wind speed is 5 miles/hour and 180 degrees (from South to North) during the execution time of 10 hours, the state of fire propagation is shown in Figure 3.5

with the ignition point of the center in the cell space (191, 133). In the figure, the black and red cells represent the burned and burning cells respectively, and all others are unburned cells. Therefore, by the graphical interface, we can clearly figure out the situation of the fire growth.

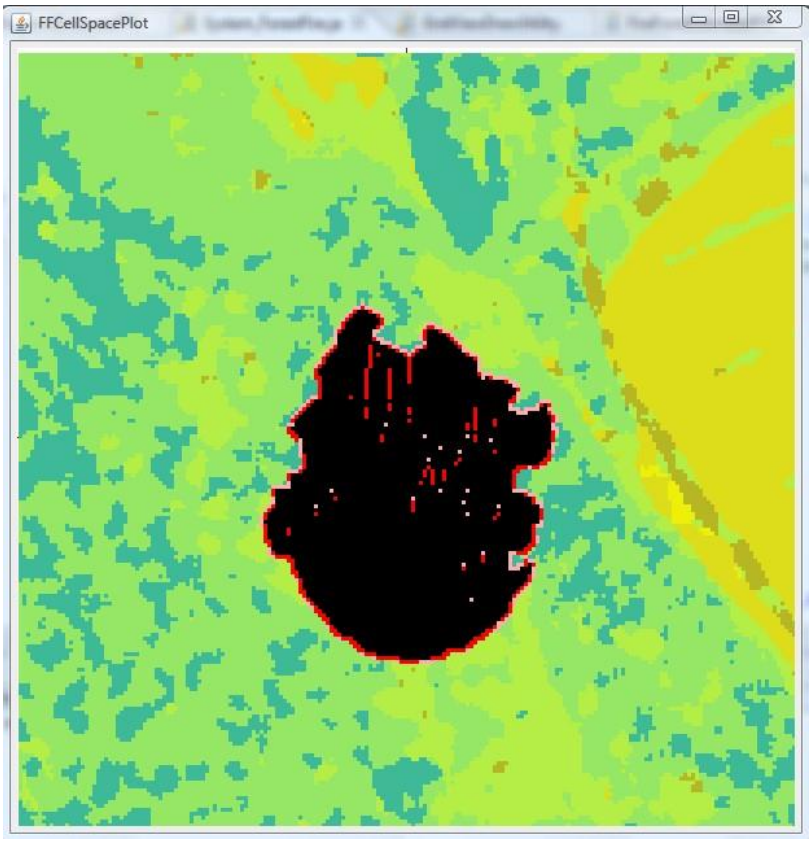


Figure 3.5 Graphical output of fire spread in DEVS-FIRE

CHAPTER 4 MEASUREMENT MODEL

The measurement model converts the output from the system model into the measurement data, which is used to compare with the real time data. In this work, we intend to estimate the evolving fire front, which represents the most important information in a wildfire spread simulation. For the measurement data, we collect the real time temperatures of distributed ground sensors in the fire field. Figure 4.1 shows how the real time data is collected on-site using ground fire sensors. In the entire cell space, a number of ground fire sensors are deployed.

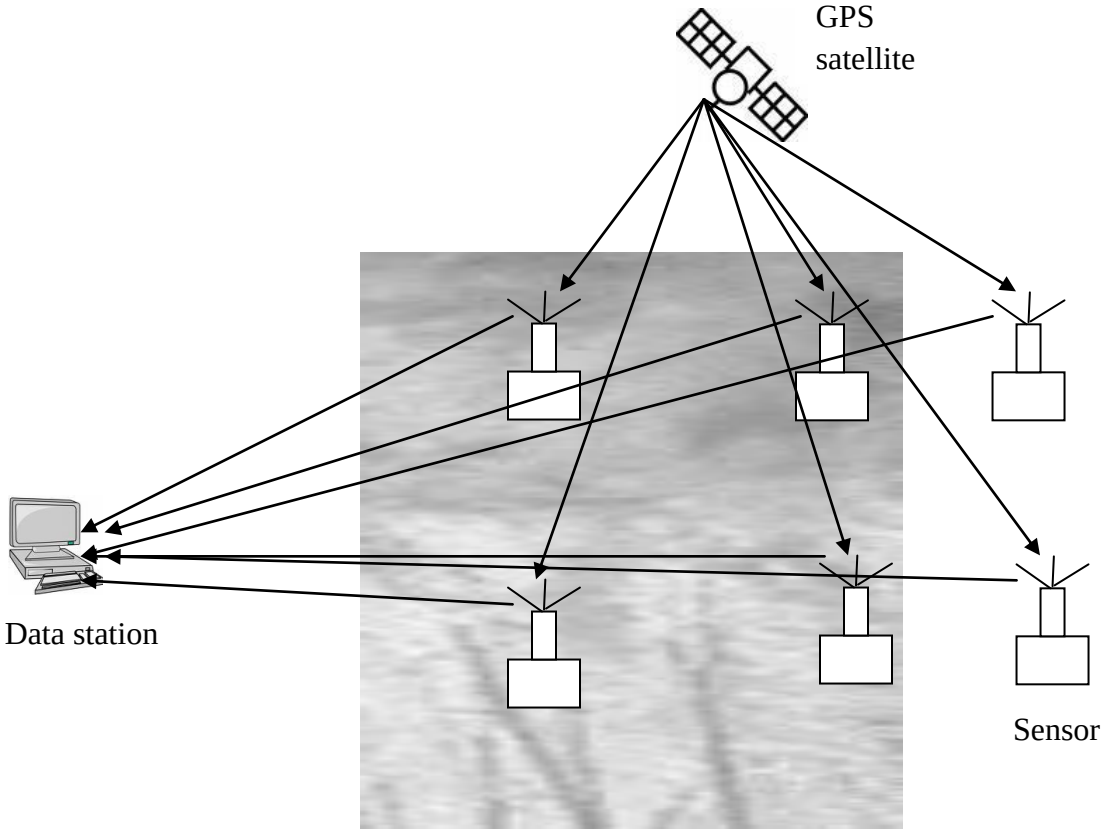


Figure 4.1 Real time data collection

Each time step, fire sensors send messages including their positions and temperature data to the data station, thus to provide real time data for fire scientists to analyze and use. Based on the collected data in this context, the measurement model is defined as follows.

The measurement model maps the system state (the fire front) to the measurement data (temperatures of deployed sensors). In DEVS-FIRE, the wildfire field is represented by a discretized cellular space where the burning cells on the fire perimeter form the fire front. Within this context, the measurement model includes two main aspects: the deployment schema of ground temperature sensors, and the function of computing sensors' temperature data from a fire front. The deployment schema defines how the sensors are deployed in the wildfire field. Examples of deployment schema include regular deployment, e.g., one sensor every 10 cells or every 20 cells, random deployment where sensors are randomly distributed in the cell space, and fire-directed deployment where more sensors are deployed around the active fire regions. These deployment schemas (and the total number of sensors) result in different locations of the sensors. This location information is used in computing the temperature data of the sensors.

4.1 Computation of temperatures

In (Kremens 2003; Mandel et al. 2008), time-temperature profile is studied and assimilated. From these researches, we can know that the temperature will shapely decrease with the increase of time and distance to the fire center as shown in Figure 4.2, which can be denoted by the formula as shown in equation (4.1).

$$T = T_c e^{\frac{-(x-x_0)^2}{\sigma^2}} + T_a, \tag{4.1}$$

where T is the temperature of the sensor; T_c refers to the temperature rise above ambient of the burning cell (°C); T_a denotes the air temperature (°C); x denotes the location of the

sensor, and x_0 denotes the location of the closest burning cell on the fire front to the sensor. Especially, Mandel et al. (2008) calibrates the relationship between the temperature and the distance to the center and conclude the related empirical values, which will be adopted in this measurement model.

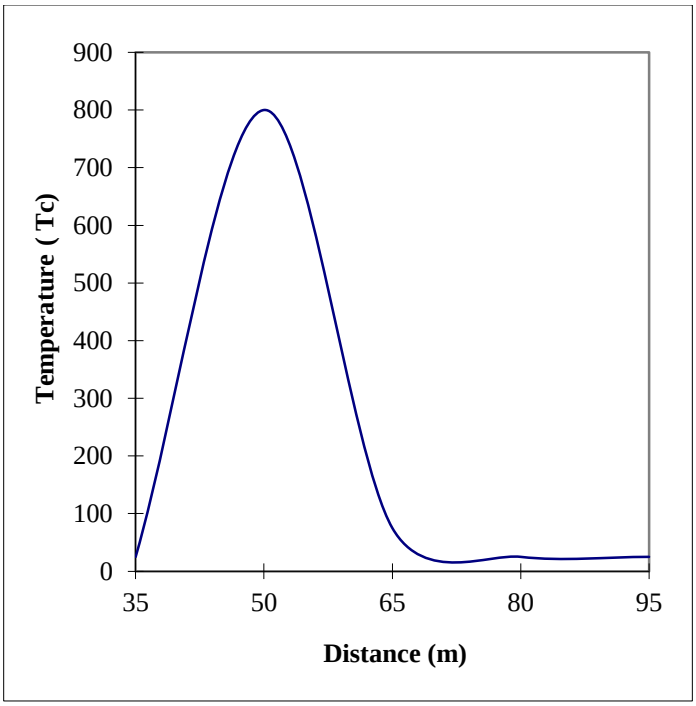


Figure 4.2 Relationship between the distance and the temperature

To compute T_c for a burning cell, equation (4.2) is used (Van Wagner 1973; Van Wagner 1975).

$$T_c = 3.9FI^{\frac{2}{3}} / h, \tag{4.2}$$

where FI is the fire intensity of the burning cell (kW m^{-1}) and h is the height above ground (m). In the DEVS-FIRE simulation, the fire intensity of a burning cell is obtained from Rothermel's Behave model. For ground temperature sensors, the height h would be their installation heights.

4.2 Example of the measurement model

To illustrate how the measurement model works to compute the temperatures of ground fire sensors, we give an example as displayed in Figure 4.3. It shows a snapshot of a fire spread simulation at some time point, when 8 cells are burning (display in red) in the 9 × 9 cell space with each cell's resolution being 15 meters. The temperature sensors are regularly deployed in the cell space, i.e. one fire temperature sensor for every three cells. In the figure, the cells that have deployed sensors are displayed in gray color. To illustrate how the temperatures are calculated, we use the formula $T = 376e^{-d^2/2\sigma^2} + 27$ (let $\sigma = 50$) to specify the relationship between the distance from the sensor to the closest burning cell and the sensor's temperature. Based on this formula, we can obtain the temperatures of all the sensors, which are {149, 267, 267, 267, 386, 386, 267, 386, 371} indexing from left to right, and top to bottom. Note that in this example, the closest distances to the fire front for these sensors are {75, 45, 45, 45, 15, 15, 45, 15, 21}.

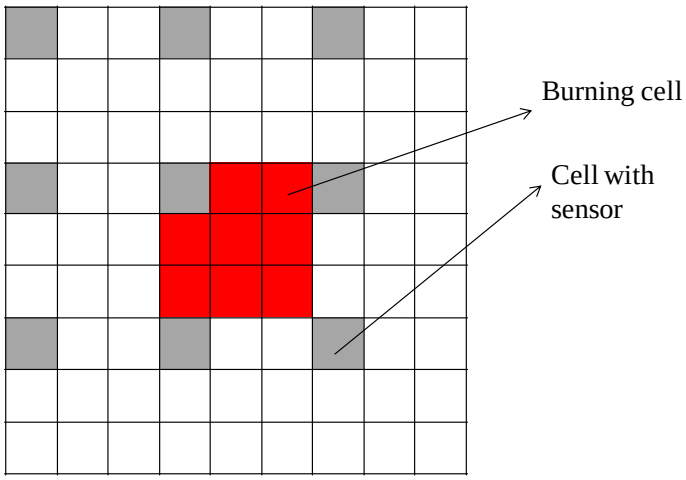
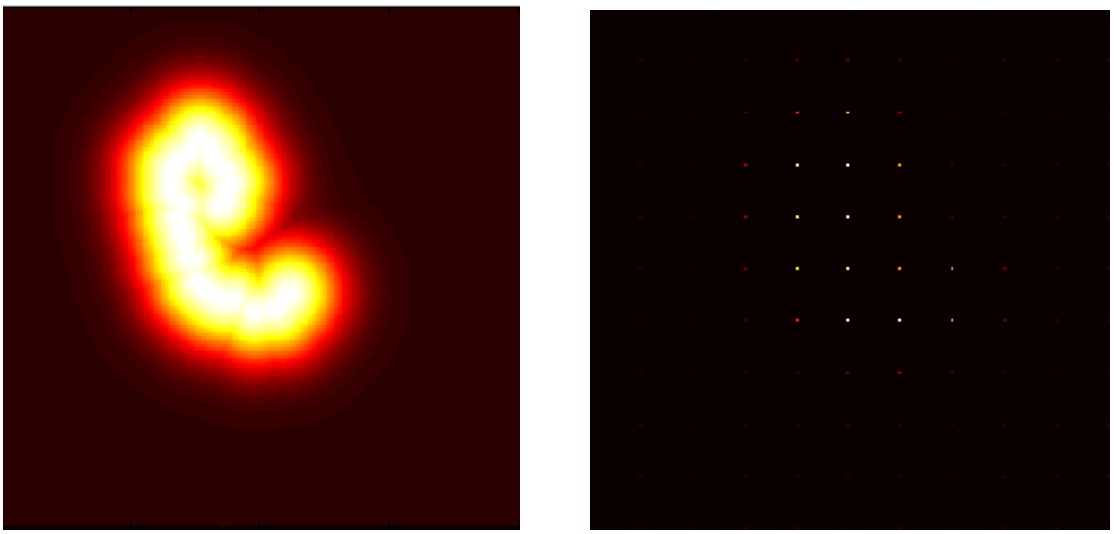


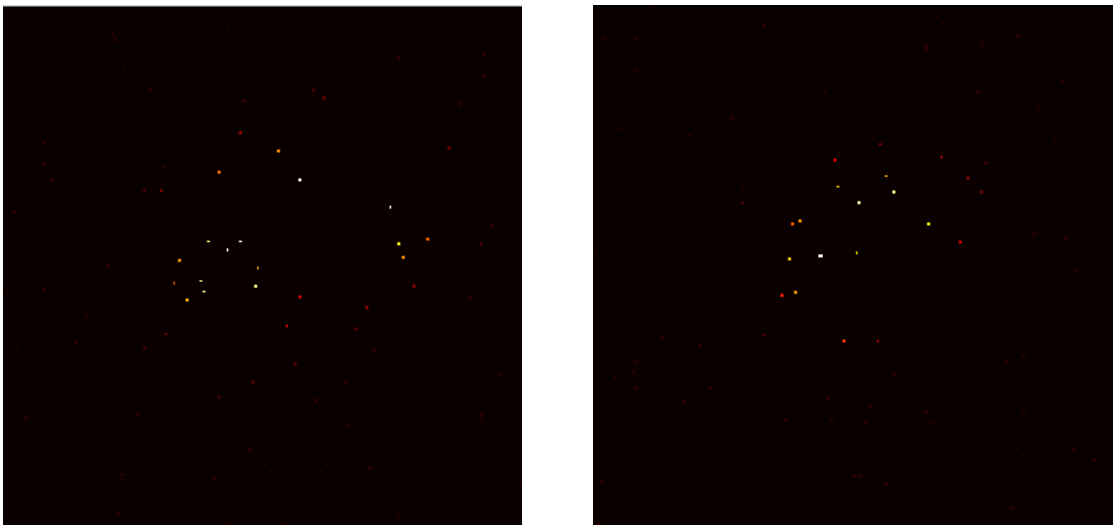
Figure 4.3 Example of the measurement model

4.3 Sensor deployment schema

To illustrate how the different deployment schema can affect the temperature data collected from sensors, Figure 4.4 shows the sensors' temperature data for a given wildfire in three different deployment schemas, in which the sensor's temperature varies smoothly from black through shades of red, orange, and yellow, to white. In the figure, Figure 4.4(a) is the real temperature map of a wildfire in a 200 × 200 cell space. This is computed by assuming each cell has a temperature sensor deployed. In reality, much smaller number of sensors will be used, and sensors will be deployed only to certain locations in the wildfire field. Figure 4.4(b), Figure 4.4(c), and Figure 4.4(d) show the temperature data for three different deployment schemas. In Figure 4.4(b), sensors are regularly deployed with one sensor every 20 cells. Overall 100 sensors are deployment in the cell space. Figure 4.4(c) and Figure 4.4(d) use the same number sensors. However, in these two cases the sensors are randomly deployed. In the figures, for the two case of 100 sensors randomly deployed, we denoted them as Temperature map 1 and Temperature map 2 respectively.



(a) (b)



(c) (d)

Figure 4.4 Temperature maps of the measurement model. (a) Real temperature map; (b) Temperature map with sensor/20 cells; (c) Temperature map 1 with 100 sensors randomly distributed; (d) Temperature map 2 with 100 sensors randomly distributed.

CHAPTER 5 DATA ASSIMILATION IN DEVS-FIRE SIMULATION

To present the data assimilation framework using SMC methods based on the DEVS-FIRE model, we first formulate the data assimilation problem for applying SMC methods. Then we present the procedure of SMC methods, including the sampling, weight computation, and resampling stages of the procedure and their associated algorithms, for assimilating data in DEVS-FIRE simulations.

5.1 Problem formulation

To improve the results of DEVS-FIRE wildfire spread simulations, the real time data from real wildfires is assimilated into the simulation model. To apply SMC methods for data assimilation, the system model of state evolution and the measurement model that maps system state to measurement data need to be defined. Based on the DEVS-FIRE simulation model, we formulate a non-linear state-space model as shown in equation (5.1).

$$\begin{cases} fire_{t+1} = DF(fire_t, t) + v_t, \\ TM_t = MM(fire_t, t) + w_t. \end{cases} \tag{5.1}$$

In equation (5.1), $fire_t$ and $fire_{t+1}$ are the system state variables of fire spread at time step t and time step t+1 respectively. In this work, we define the system state variable as the evolving fire front, which is the most important information in a wildfire spread simulation. Specifically, the fire front at time step t ($fire_t$) is composed of all the burning cells along the fire perimeter at time step t. These cells ignite their unburned neighbors in the wildfire spread simulation. They represent the initial condition for the simulation to proceed to the next time step t+1. In our implementation, the fire front is specified by a vector that includes the (x, y) IDs of all the cells on the fire front. TM_t is the measurement variable. In this work, we consider the measurement variables as the temperature data

obtained by ground temperature sensors deployed in the fire field. It is specified by a vector including ($sensor_i$, $tempearture_i$) pairs for all the sensors, where $sensor_i$ denotes the ID for the ith sensor and $tempearture_i$ denotes the temperature data of $sensor_i$ at the current time step. Note that we assume the total number of sensors and their locations are pre-defined – these are the parameters of the measurement model. *DF* is the system model that defines the evolution of system state over time. This is the DEVS-FIRE simulation model in our work. Given a fire front at time step t, through DEVS-FIRE simulation to the next time point we can obtain the fire front at time step t+1. *MM* is the measurement model that maps the system state variable (the fire front) to the ground temperature sensors' temperature data. It defines how to compute sensor data for all the temperature sensors from a given fire front. The v_t and w_t refer to the noises of the system state and those of the measurement data respectively.

Both the system model and the measurement model are essential components of the data assimilation system. The system model is the DEVS-FIRE simulation model, which has been described in Chapter 3. The slope, aspect, fuel model, and weather data used in computing fire spread behavior are considered as parameters of the DEVS-FIRE model. Based on these parameters and the simulation mechanism, DEVS-FIRE computes the evolution of the fire front over time. The measurement model has been described in Chapter 4.

5.2 Data assimilation using SMC methods

Having formulated the problem and described the system model and measurement model, in this section we present the SMC methods for assimilating temperature sensor data in DEVS-FIRE wildfire simulations. The SMC methods used in this work

implement the sequential importance sampling with resampling (SISR) principle described in section 5.2.1. Figure 5.1 shows the structure of SMC methods and the procedure of the data assimilation algorithm. In the figure, the rectangle boxes represent the major activities in one time step of the algorithm, and the circles represent the data/variables. The data assimilation algorithm runs in a stepwise fashion. At time step t, the set of system state variables, i.e., the fire fronts, from time step t-1 (denoted as S_{t-1} in Figure 5.1) are fed into the system model, which is the simulation model of DEVS-FIRE. Based on the DEVS-FIRE fire spread simulation, these fire fronts evolves to a set of new fire fronts (denoted as S'_t). To compute the importance weights of these new simulated fire fronts, the temperature sensor data need to be computed from these fire fronts and compared with the real sensor data (the real observation, denoted as m_t in Figure 5.1). Thus for each fire front in S'_t, its temperature data are computed according to the locations of the temperature sensors and the measurement model as described in Chapter 4. The set of measurement data for all fire fronts are denoted as M'_t. By comparing the measurement data with the real temperature sensor data, the importance weight of each fire front is calculated and normalized. These weights are used by the resampling algorithm that draws a set of offspring samples from S'_t with probability proportional to the importance weights. The set of resampled fire fronts are denoted as S_t and are used as the inputs for the next step t+1. We note that in Figure 5.1, the time step t-1, t, and t+1 are used to indicate the stepwise fashion of the algorithm. The actual time interval between two consecutive steps is usually defined by how often the sensor data is collected, for example, in every 20 minutes.

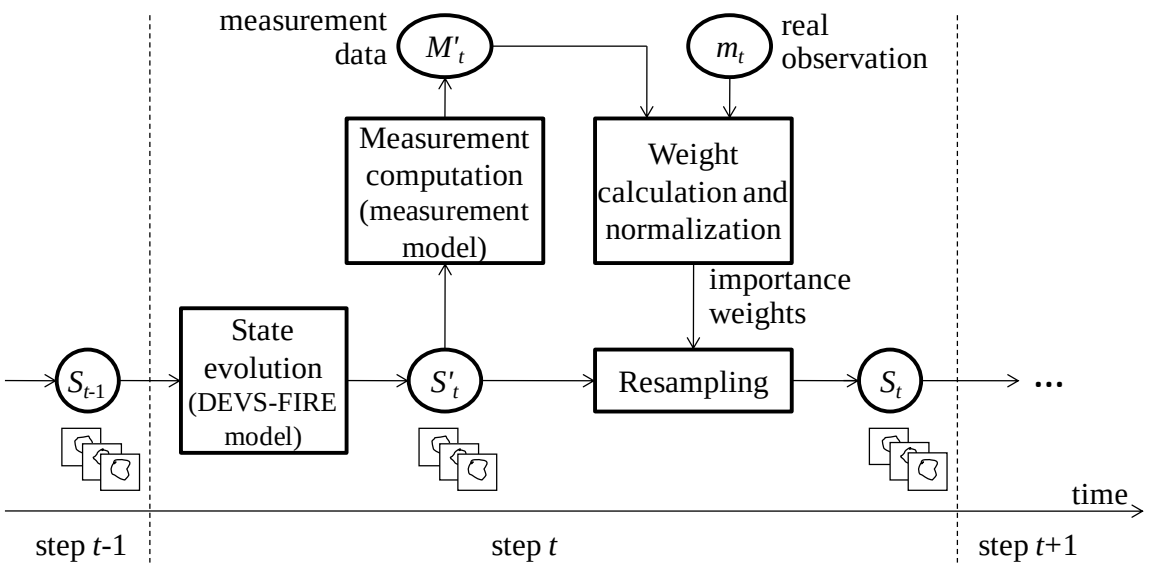


Figure 5.1 Data assimilation based on SMC methods

The algorithm of SMC methods that implement the above procedure is given below. In the algorithm, the set of fire fronts is represented by a set of particles, where each particle is a fire front. The algorithm starts by initializing N particles representing the initial fire fronts when the fire is ignited (we start the data assimilation from when the fire is ignited). Then the algorithm goes through multiple iterations, each of which includes the sampling, weight computation, and resampling stages. At the sampling stage, all the particles go through the DEVS-FIRE simulation to obtain their corresponding new fire fronts (with noises added) of the next time step. At the weight computation stage, the differences between the temperature data (computed using the measurement model) from these fire fronts and the real temperature sensor data from the real wildfire are calculated. These temperature differences decide the importance weights of the particles. Then the resampling stage selects the fire fronts based on their normalized importance weights to

form a new set of particles, which are used in the next iteration of the algorithm. Below we describe the sampling, weight computation, and resampling stages in detail.

Figure 5.2 illustrates how the data assimilation works for estimating fire front by assimilating ground temperature sensor data. In the figures, the top left window is the "real" wildfire, which is unknown to the data assimilation system and needs to be estimated. Here the "real" fire is generated by a simulation (see the identical-twin experiment description in Chapter 6 for more details). The top right window is the temperature map of the fire area due to the "real" fire front. In this window, the blue dots indicate the locations of the ground temperature sensors. The bottom right window shows the temperature sensor data collected from the temperature sensors. These data will be assimilated by the SMC method for estimating the "real" fire front. The bottom left window shows the data assimilation results. In this window, the red perimeter represents the real fire. It is displayed in this window for comparing with the data assimilation results. The gray perimeters represent different estimations (totally 210 estimations because 210 particles were used in this example) of the fire fronts. By displaying all these estimations together, one can treat the shades of a cell as its burning possibility. The darker the shade, the higher possibility that the cell is at the fire front is.

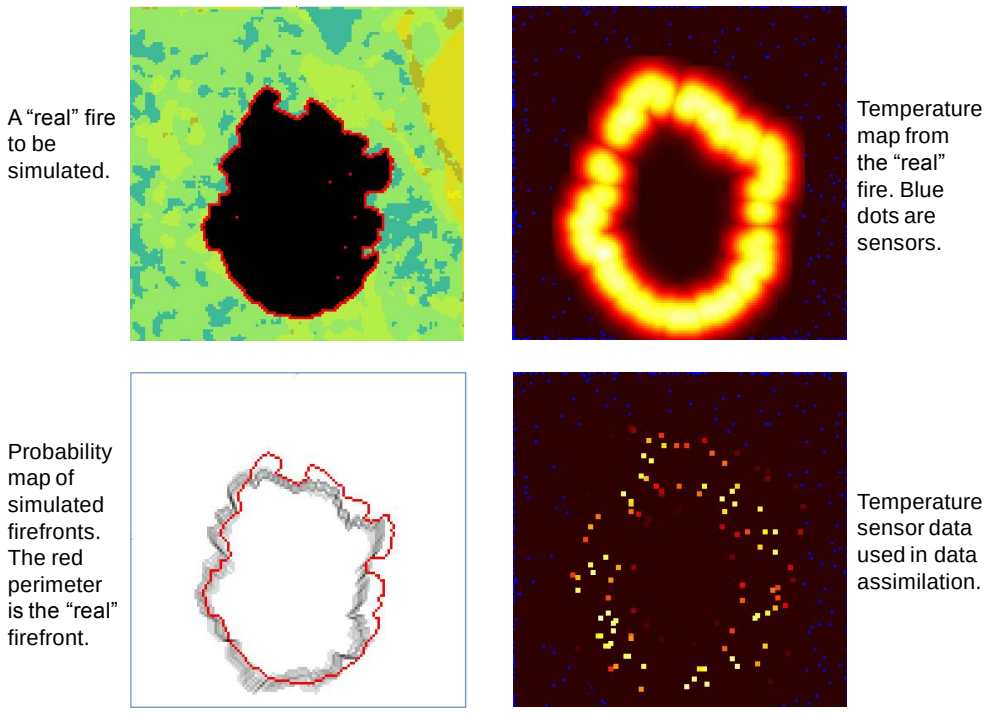


Figure 5.2 Fire front estimation by assimilating ground temperature sensor data

5.2.1 Sampling using DEVS-FIRE simulation

The sampling stage evolves all the particles (the fire fronts) to the next time point using the wildfire spread simulation model. Specifically, for each particle we create a DEVS-FIRE simulation starting from the fire front of that particle and run the simulation for a given time period to obtain a new fire front. The simulation time period is determined by how often the sensor data is collected, such as, every 20 minutes. We note that the DEVS-FIRE model is a deterministic wildfire spread simulation model. Thus if two particles have the same fire front shapes, after the simulations they will end with exactly the same simulation results, that is, new fire front shapes. In order to provide stochastic state evolution behavior required by SMC methods, after the DEVS-FIRE simulation we add graph noises to the generated fire front shapes.

Algorithm of SMC methods

1. Initialization. Initialize the particles $\{fire_0\}_{i=1}^{N}=0$ and set t=1.
2. Sampling. Evolve the particles to $fire_{t+1}^{(i)}, i=1,2,...N$. This includes two steps. First, run the DEVS-FIRE simulation from the current samples (fire fronts) to generate new samples as the new fire fronts. Second, add noises to these samples.
3. Weight computation. Calculate importance weights $\{q_t^{(i)}\}_{i=1}^{N}$ by $q_t^{(i)} = p(TM_t \mid fire_{t-1}^{(i)}),\ i=1,2,...N,$ then normalize them $\tilde{q}_t^{(i)} = q_t^{(i)} / \sum_{j=1}^{N} q_t^{(j)}$.
4. Resampling. Draw the new particles to replace the old ones according to $\Pr(fire_t^{(i)} = fire_t^{(j)}) = \tilde{q}_t^{(j)}, i=1,2,..,N.$ Set $t \leftarrow t+1$ and go to step 2.

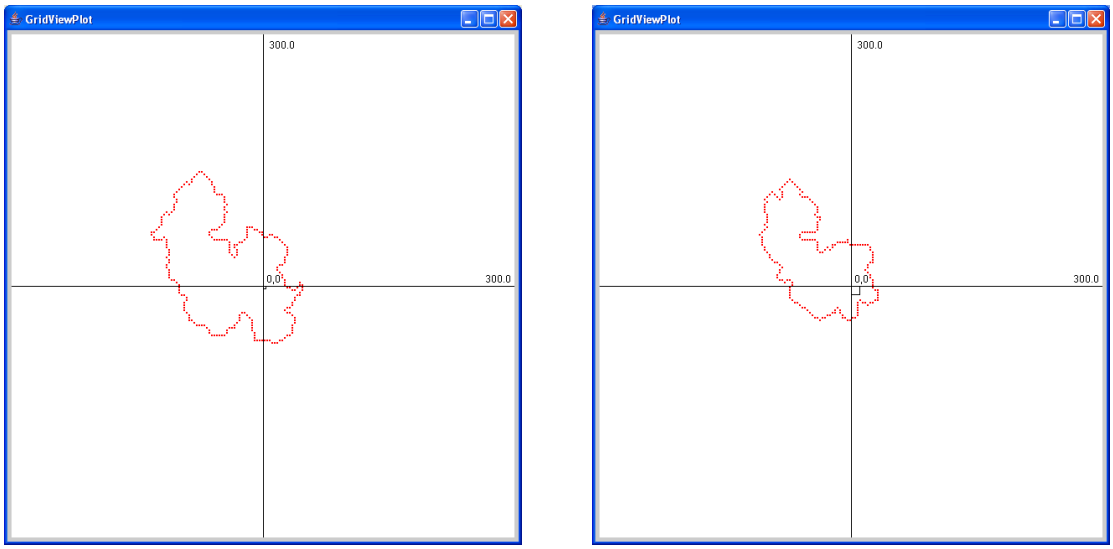
Given a fire front shape, the algorithm that adds graph noise for generating a new fire front is described below. First, we divide the fire front into multiple segments, each of which consists of an equal number of burning cells. To add graph noise to a segment we introduce a noise range denoted as *rd*. The noise range *rd* defines the range of change (in number of cells) inside or outside a cell along the direction from this cell to the ignition point. Different segments have different noise ranges, but all cells of a segment share the same noise range. Next, for every cell in a segment, a new cell is randomly selected in the noise range of [*-rd*, *rd*] to replace the original cell. We continue this for all the cells on the fire front and finally re-connect all the new cells to form a new continuous fire front. The algorithm of generating new fire front by adding graph noises is given below.

This algorithm adds variances to the fire fronts resulting from the fire spread simulations. To illustrate the effect of the algorithm, Figure 5.3 displays three generated fire fronts for a given fire shape using graph noises. Figure 5.3(a) is the original fire front,

based on which Figure 5.3(b), Figure 5.3(c), and Figure 5.3(d) are three generated fire fronts with graph noises added.

Algorithm of generating fire front with graph noises added

1. Divide the fire front into n consecutive segments, denoted as SEG_1, SEG_2, …, SEG_n.
2. Randomly generate noise ranges rd_1, rd_2, …, rd_n of all the segments SEG_1, SEG_2, …, SEG_n.
3. For every burning cell c in segment SEG_i, randomly choose a new cell in the noise range [$-rd_i$, $+rd_i$] of the original cell along the direction from the ignition point to the cell. Set the new cell as *burning* and the original cell as *unburned*.
4. Scan the new burning cells, and construct a continuous fire front.



(a) (b)

(c) (d)

Figure 5.3 Fire fronts with random noises. (a) Original fire front; (b), (c), (d) Three generated fire fronts with random noises.

5.2.2 Weight computation

To evaluate how well a simulated fire front (a particle) matches the real wildfire, we need to compare the temperature sensor data computed from the simulated fire with the real sensor data collected from the real wildfire, and assign a weight to the particle. The weight computation stage computes the importance weights for all the particles. This is done in two steps: 1) measurement data computation, and 2) weight calculation and normalization (see Figure 5.1). In measurement data computation, we compute the temperature sensor data from each particle based on the particle's fire front. Let K denote the total number of sensors, se_i denote the ith sensor ($i \in \{1,2,\ldots,K\}$). Based on a particle's fire front, all sensors' locations, and the measurement model described in Chapter 4, each sensor's temperature data is computed. The set of temperature data for all the sensors is denoted as $m' = \{temperature_i'\}$, where $temperature_i'$ is the temperature data for se_i. We note

that *m*' computed from different particles will have different values because of particles' different fire front shapes. After *m*' is computed, to account for sensor noises that exist in real sensors, we add random noises to each sensor's temperature data. The range of noise is pre-defined by the user and is determined according to the noises of real sensors. The set of temperature data with noises added is denoted as $m = \{temperature_i\}$. This is the measurement data and is computed for every particle.

In the weight calculation step, the temperature data *m* of each particle is compared with the real observation data (denoted as $m^{real} = \{temperature_i^{real}\}$) to compute the importance weights of the particles. Below is the algorithm of computing a particle's weight based on its measurement data *m* and the real observation data m^{real}. The first step of the algorithm calculates each sensor data's contribution and returns a value between 0 and 1 based on how much difference exists between the measurement data and the real observation data. This is done through the equation $ps_i = e^{-d_i \times d_i / 2\sigma^2}$, where d_i is the difference between the measurement data and the real observation data for sensor se_i. As can be seen, the larger the temperature difference, the smaller the contribution is. In the ideal case, if there is no difference between a sensor's measurement data and the corresponding real observation data, the contribution of that sensor equals to 1.0. Step 1.c adjusts the contribution computed from step 1.b. We add this adjustment step because the contribution computed in step 1.b is always a positive value (even if there is large difference between $temperature_i$ and $temperature_i^{real}$). Knowing that a large difference between $temperature_i$ and $temperature_i^{real}$ is an undesirable situation and should not have any positive contribution to the importance weight of the particle, step 1.c adjusts the

contribution – it returns a negative value if ps_i is smaller than a pre-defined threshold a and a positive value if ps_i is larger than a. Figure 5.4 shows how the adjusted contribution is calculated for a=0.3. After step 1, step 2 computes the weight by summing up the adjusted contributions for all the sensors and uses it to calculate the weight of the particle. Finally, after all particles' importance weights are calculated, they are normalized (not shown in the algorithm below). The normalized importance weights are used in the resampling step for selecting particles for the next iteration of the data assimilation.

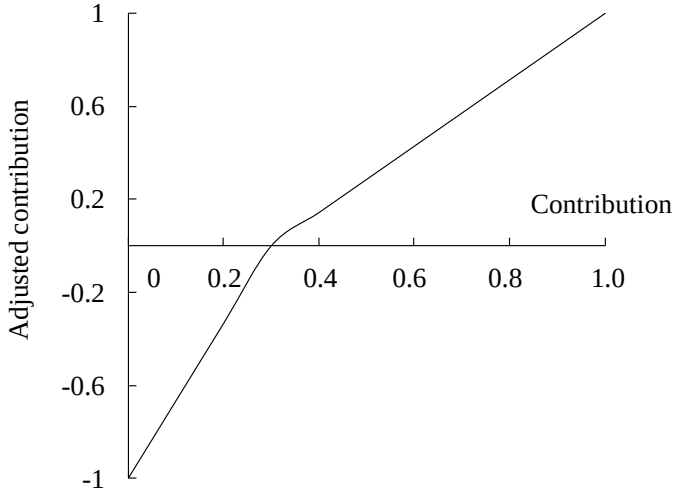


Figure 5.4 Adjusted contribution computation (a=0.3)

5.2.3 Resampling algorithm

We use the multinomial resampling to implement our resampling algorithm. The algorithm is described below, where $wt_t^{(i)}$ is the importance weight of the ith particle at time step t, and N is the total number of particles. Firstly, the cumulative sums of the normalized weights of N particles $(\tilde{q}_t^{(1)}, \tilde{q}_t^{(2)}, ..., \tilde{q}_t^{(i)}, ..., \tilde{q}_t^{(N)})$ are computed, where $\tilde{q}_t^{(i)} = wt_t^{(i)} / \sum_{j=1}^{N} wt_t^{(j)}$. Then N random numbers (denoted as u_k) are generated between 0

and 1. Finally, we count the number of u_k that fall into the interval of the cumulative sum of $\tilde{q}_t^{(i-1)}$ and that of $\tilde{q}_t^{(i)}$. This number decides how many copies of the *i*th particle will be selected. Figure 5.5 shows the case with 5 fire fronts, and the regenerated fire fronts are $\{fire^1, fire^1, fire^2, fire^4, fire^5\}$.

Algorithm of computing a particle's weight based on its measurement data and the real observation data

---

1. For each sensor se_i,
   a. compute the temperature difference between $temperature_i$ and $temperature_i^{real}$: $d_i = temperature_i - temperature_i^{real}$
   b. Compute the contribution $ps_i = e^{-d_i \times d_i / 2\sigma^2}$ for se_i
   c. calculate the adjusted contribution $ps_i' = \begin{cases} (ps_i - a)/(1-a), ps_i > a \\ (ps_i - a)/a, else \end{cases}$
2. For all the sensors $\{se_i\}_{i=1}^K$ in the cell space, calculate the weight $wt = e^{\sum_{i=1}^{K} ps_i'}$. This is the importance weight of the particle.

Algorithm of the multinomial resampling

---

1. Compute the cumulative sums of the normalized weights of *N* particles ($\tilde{q}_t^{(1)}$, $\tilde{q}_t^{(2)}$, …, $\tilde{q}_t^{(i)}$, …, $\tilde{q}_t^{(N)}$), where $\tilde{q}_t^{(i)} = wt_t^{(i)} / \sum_{j=1}^{N} wt_t^{(j)}$.
2. Generate *N* ordered random numbers $\{u_k\}_{k=1}^N$, where $u_k \in (0,1)$.
3. The new generated samples are composed of n_i copies of $fire^{(i)}$, where n_i is the number of $u_k \in (\sum_{j=1}^{i-1} \tilde{q}_t^{(j)}, \sum_{j=1}^{i} \tilde{q}_t^{(j)})$.

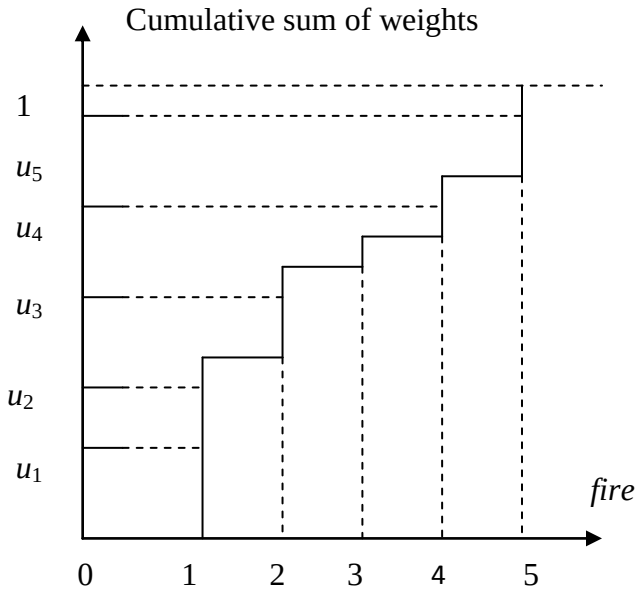


Figure 5.5 Multinomial resampling. The generated numbers of $fire^1$ to $fire^5$ are 2, 1, 0, 1, 1.

5.3 Software architecture

In the SMC methods algorithm, we run number of particles simulations for each step because of the introduction of the uncertainty. This will greatly increase the computations of the system. The number of particles, in some extent, decides the precision of the state estimation. Especially for the dynamic state prediction of DEVS-FIRE, DEVS-FIRE itself is a simulation, which consumes a lot of resources from the aspects of time and space. In this application, we adopt client-server computing to improve the performance. Figure 5.6 displays the structure of client-server computing.

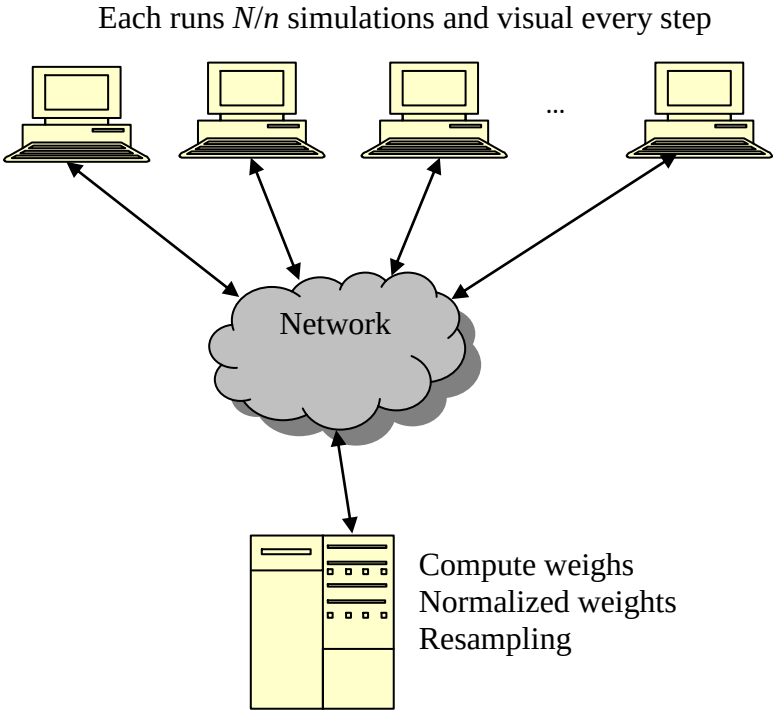


Figure 5.6 Computational architecture

From the figure, we can see multiple clients and the server are connected by network (LAN). For each step of the SMC methods algorithm, the server distributes *N* fire shape with noises to *n* client, and the clients run simulations for the period of time after receiving the corresponding particles. When clients finish the simulations, they send the results, which are a series of fire fronts and intensities of each cell, to the server for processing. Finally, the sever computes weights, normalizes weights, and does resampling for future use after receiving all the simulation results from clients. The pseudo codes for both client and server are shown as follows.

Server side

```
for i=0 to N-1
   send(fire(i, k), client(i/n));
for i=0 to N-1
   receive(fire(i, k+1), client(i/n));
   computeWeights(N, k+1);
   normalizeWeights(N, k+1);
   resampling(N, k+1);
```

Client side

```
for i=0 to N/n-1
   receive(fire(clientId*N/n+i, k), server);
   fire(clientId*N/n+i,k+1)=
      DEVSFIRE(s,a,f,w,fire(clientId*N/n+i, k), t);
   send(fire(clientId*N/n+i, k+1),server);
```

CHAPTER 6 EXPERIMENTAL RESULTS

6.1 Experimental methods and designs

We used the identical-twin experiment, widely used in data assimilation research, to evaluate the data assimilation system of DEVS-FIRE. The purpose of identical-twin experiments is to study the assimilation in ideal situations and evaluate the proximity of the prediction to the true states in a controlled manner. In the identical-twin experiment, a simulation is first run, and the corresponding data is recorded. These simulation results are considered as "true"; therefore, the observation data obtained here is regarded as the real observation data (because they come from the "true" model). Consequently, we estimate the system states from the observation data using SMC methods, and then check whether these estimated results are close to the "true" simulation results. In this section, we use three terms: "real" fire, filtered fire, and simulated fire, to help us to present the experimental results. A real fire is the simulation from which the real observation data is obtained. A simulated fire is the simulation based on some "error" data ("error" in the sense that the data are different from those used in the real fire), for example, imprecise weather data. This is to represent the fact that wildfire simulations usually rely on imperfect data as compared to real wildfires. Finally, a filtered fire is the data assimilation-enhanced simulation based on the same "error" data as in the simulated fire. In our experiments, we intended to show a filtered fire gave more accurate simulation results by assimilating observation data from the real fire even it still used the "error" data.

The differences between a real fire and a simulated fire are due to the imprecise data such as wind speed, wind direction, GIS data, and fuel model, used in the simulation. In

our experiments, we chose to use the incorrect wind conditions (wind speed and wind direction) as the "error" data. Table 6.1 shows the configurations of four sets of experiments. The real wind speed and direction are 8 (miles/hour) and 180 degrees (from South to North) with random variances added every 10 minutes. The variances for the wind speeds are in the range of –2 to 2 (miles/hour) (denoted as 8±2 in the table), and the variances for the wind direction are in the range of -20 to 20 (degrees) (denoted as 180±20 in the table). Our first two cases introduced errors to the wind speeds and made the wind directions to be exactly the same as the real wind direction. In case 1 the wind speed was randomly generated based on 6 (miles/hour) with variances added in the range of –2 to 2 (miles/hour). In case 2, the wind speed was randomly generated based on 10 (miles/hour) with variances added in the range of –2 to 2 (miles/hour). Our next two cases introduced errors to the wind directions only: case 3 used wind direction of 160 with added variances in the range of ±30 degrees; case 4 used wind direction of 200 with added variances in the range of ±30 degrees. For all the implementations, all other parameters including GIS data and the fuel data are same (the cell space is 200 × 200, and the cell size is 30 m), and the real time data (ground sensors' temperatures) are available every 20 minutes.

Table 6.1 Experimental sets of wind factor

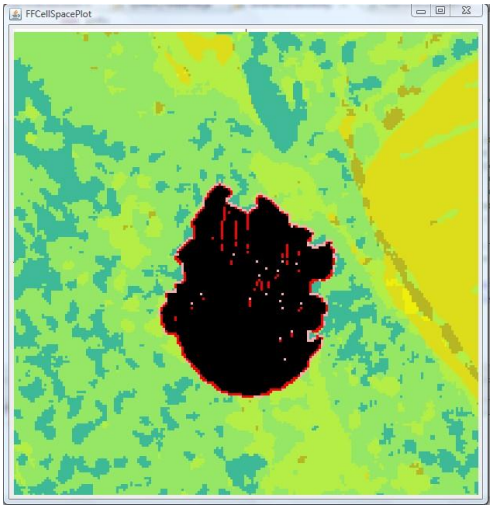
<table>
<tr><th rowspan="2">Cases</th><th colspan="2">"Error" data</th><th colspan="2">Real data</th></tr>
<tr><th>Speed (miles/hour)</th><th>Direction (degrees)</th><th>Speed (miles/hour)</th><th>Direction (degrees)</th></tr>
<tr><td>1</td><td>6±2</td><td rowspan="2">No errors</td><td rowspan="4">8±2</td><td rowspan="4">180±20</td></tr>
<tr><td>2</td><td>10±2</td></tr>
<tr><td>3</td><td rowspan="2">No errors</td><td>160±30</td></tr>
<tr><td>4</td><td>200±30</td></tr>
</table>

6.2 Experimental results

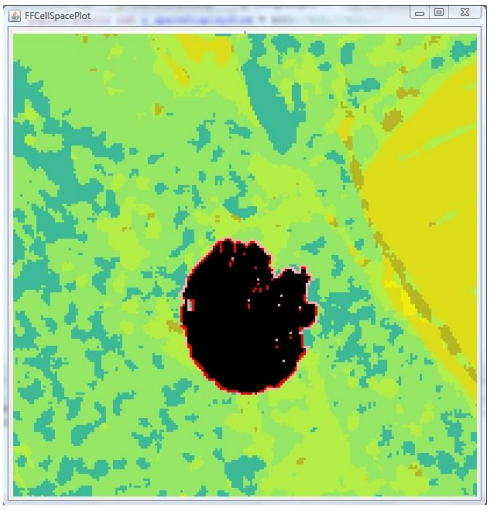
6.2.1 Wind speed

We implemented data assimilation in DEVS-FIRE with imprecise data as shown in Table 6.1. The fire spreads (run DEVS-FIRE) with the real data and the error data are displayed in Figure 6.1 (the burning cells and the burned cells are displayed in red and black respectively). In the figures, Figure 6.1(a) displays the real fire after 18 steps (20 minutes each step), which is used to generate the observation data for case 1 ~ 4. Figure 6.1(b) and Figure 6.1(c) show the simulated fires for case 1 and case 2 respectively.

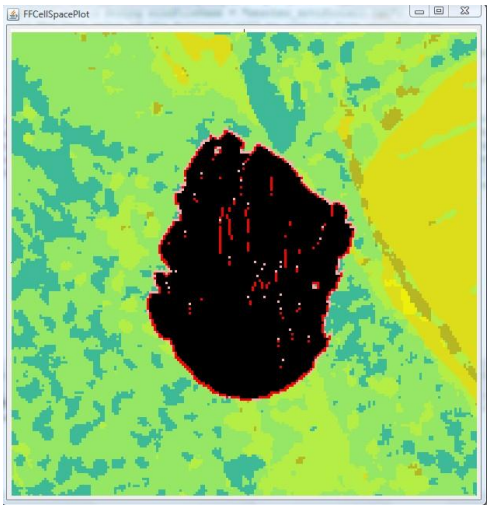
By assimilating the observation data into DEVS-FIRE, the filtered fires are obtained as shown in Figure 6.2. Figure 6.2(a) and Figure 6.2(b) display the filtered fires (the fire fronts are displayed in blue) for case 1 and case 2, compared with the real fire (the fire front is displayed in red) and the simulated fires (the fire fronts are displayed in green). From these figures we can see that after applying data assimilation to the DEVS-FIRE, more correct results are obtained. Figure 6.3 displays perimeters (case 1 in Figure 6.3(a) and case 2 in Figure 6.3(c)) and areas (case 1 in Figure 6.3(b) and case 2 in Figure 6.3(d)) of the real fire, the simulated fire, and the filtered fire for case 1 and case 2 from time step 1 to 18. From the figures, we know that the filtered fires are closer to the corresponding real fire than the simulated fires for both of case 1 and case 2.



(a)

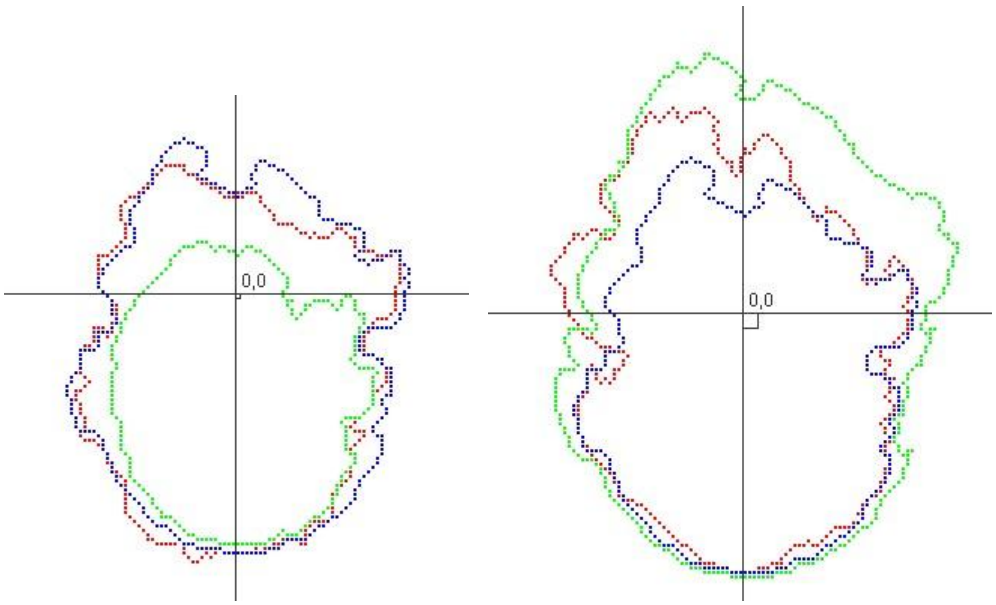


(b)



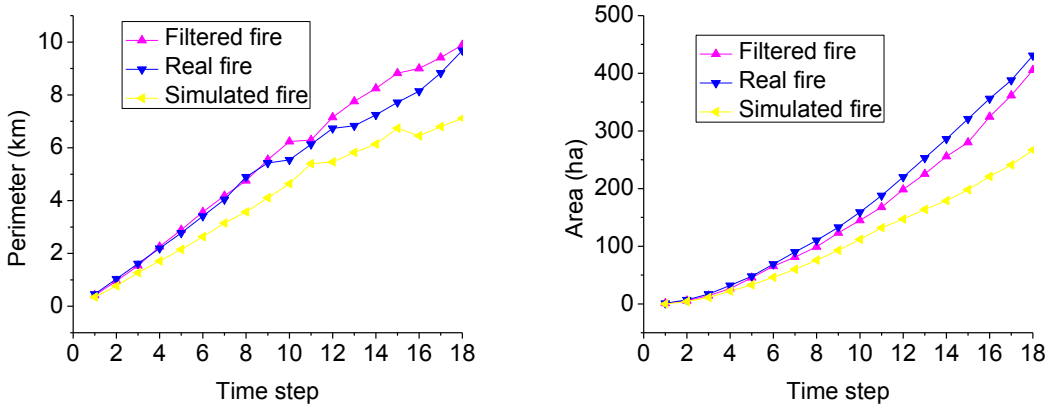
(c)

Figure 6.1 Real fire and simulated fires for case 1 and case 2. (a) Real fire after 360 minutes; (b) Simulated fire for case 1 after 360 minutes; (c) Simulated fire for case 2 after 360 minutes.

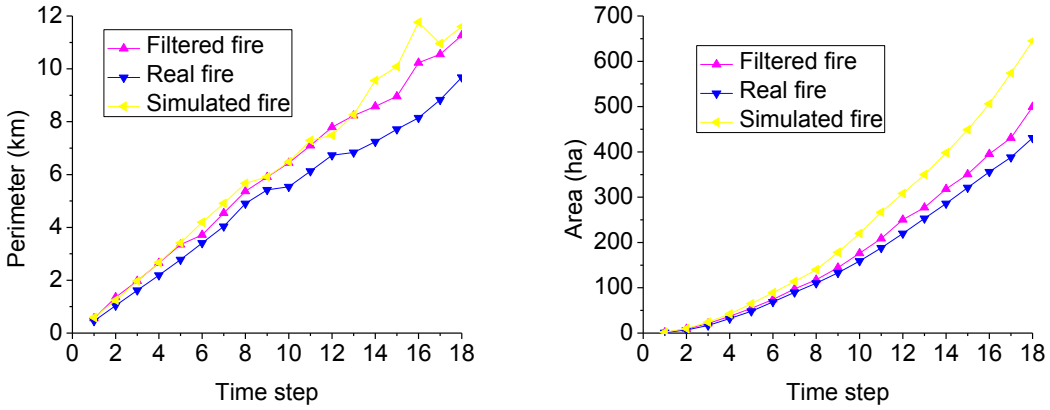


(a) (b)

Figure 6.2 Comparisons of real fire, simulated fires, and filtered fires for case 1 and case 2. (a) Case 1; (b) Case 2.



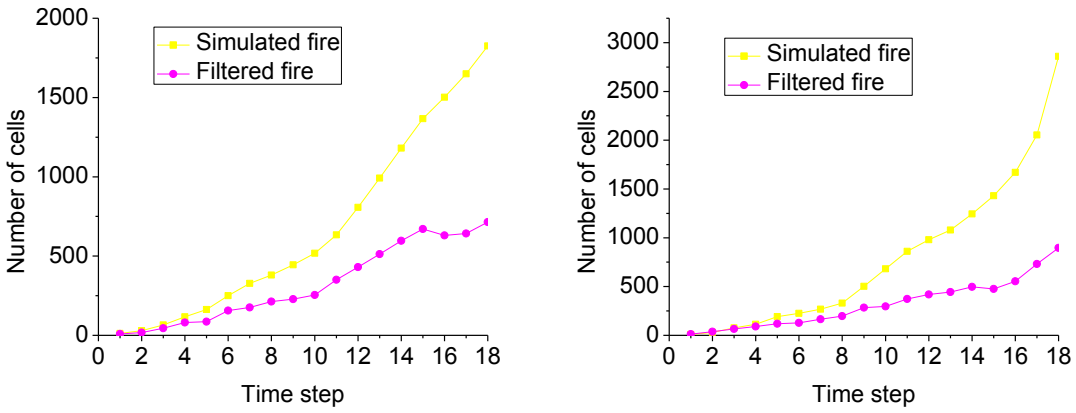
(a) (b)



(c) (d)

Figure 6.3 Perimeters and areas of real fire, simulated fires, and filtered fires for case 1 and case 2. (a) Perimeters for case 1; (b) Areas for case 1; (c) Perimeters for case 2; (d) Areas for case 2.

Symmetric set difference, a set of members in either set, but not in both, can be another metric to measure the similarity between two fire fronts. The smaller the symmetric set difference, more similar two graphics are. The value of symmetric set difference of two same fire fronts is 0. We also use it to compare the differences between the real fire and the filtered fire (simulated fire). Figure 6.4 displays the symmetric set differences for case 1 (Figure 6.4(a)) and case 2 (Figure 6.4(b)) from time step 1 to 18. In the figures, the horizontal axis and the vertical axis represent the time step and the number of burning cells in the cell space respectively. From the graphs, we know that the symmetric set difference between the real fire and the filtered fire (display in magenta) is smaller than that between the real fire and the simulated fire (display in yellow) at time step 18. Therefore, the filtered fires are more "correct" than the simulated fires. Moreover, case 1 has more improvements than case 2 because it significantly reduces the symmetric set difference by applying SMC methods. This also can be reflected in Figure 6.2.



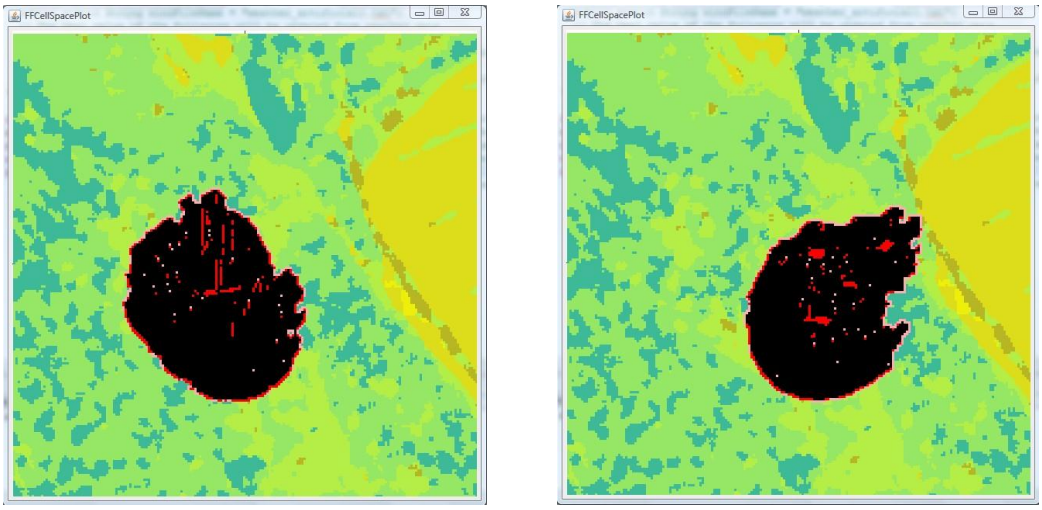
(a) (b)

Figure 6.4 Symmetric set differences for case 1 and case 2. (a) Case 1; (b) Case 2.

6.2.2 Wind direction

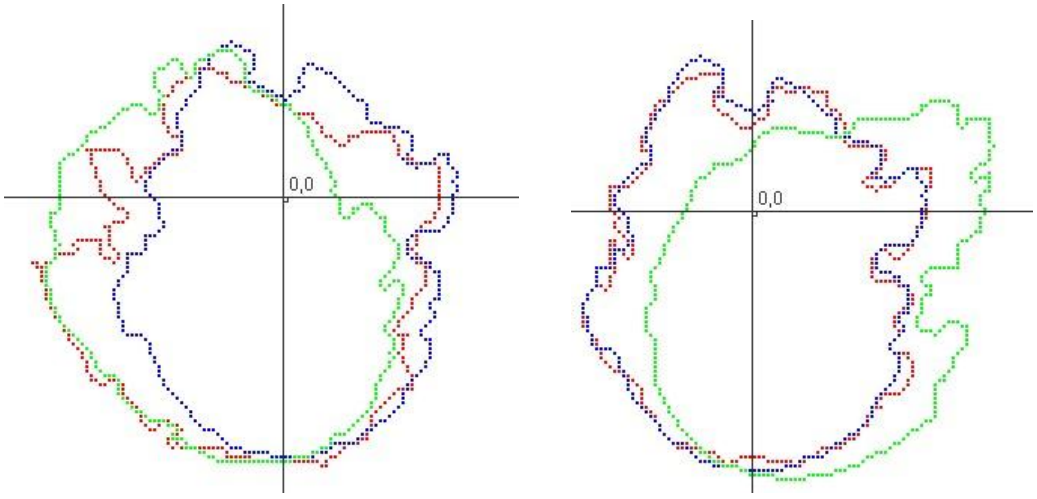
For case 3 and case 4 in Table 6.1, the wind directions have errors with respect to the real wind condition. Figure 6.5(a) and Figure 6.5(b) show the simulated fires (the burning cells and the burned cells are displayed in red and black respectively) for case 3 and case 4 after 18 time steps (20 minutes for each step). Figure 6.6(a) and Figure 6.6(b) display the filtered fires (the fire fronts are displayed in blue) for case 3 and case 4, compared with the real fires (the fire fronts are displayed in red) and the simulated fires (the fire fronts are displayed in green). Figure 6.7 displays perimeters (case 3 in Figure 6.7(a) and case 4 in Figure 6.7(c)) and areas (case 3 in Figure 6.7(b) and case 4 in Figure 6.7(d)) of the real fire, the simulated fires, and the corresponding filtered fires for case 3 and case 4. From all these graphs, we know that better estimations are also obtained by applying data assimilation in DEVS-FIRE. Figure 6.8 shows the symmetric set differences for case 3 (Figure 6.8(a)) and case 4 (Figure 6.8(b)) from time step 1 to 18. From the graphs, we know that the symmetric set difference between the real fire and the filtered fire (display in magenta) is much smaller than that between the real fire and the

simulated fire (display in yellow) for both case 3 and case 4. Therefore, the fire spread predictions are greatly improved. These results are also coincident with those in Figure 6.6.



(a) (b)

Figure 6.5 Fire spreads for case 3 and case 4. (a) Simulated fire for case 3 after 360 minutes; (b) Simulated fire for case 4 after 360 minutes.



(a) (b)

Figure 6.6 Comparisons of real fire, simulated fires, and filtered fires for case 3 and case 4. (a) Case 3; (b) Case 4.

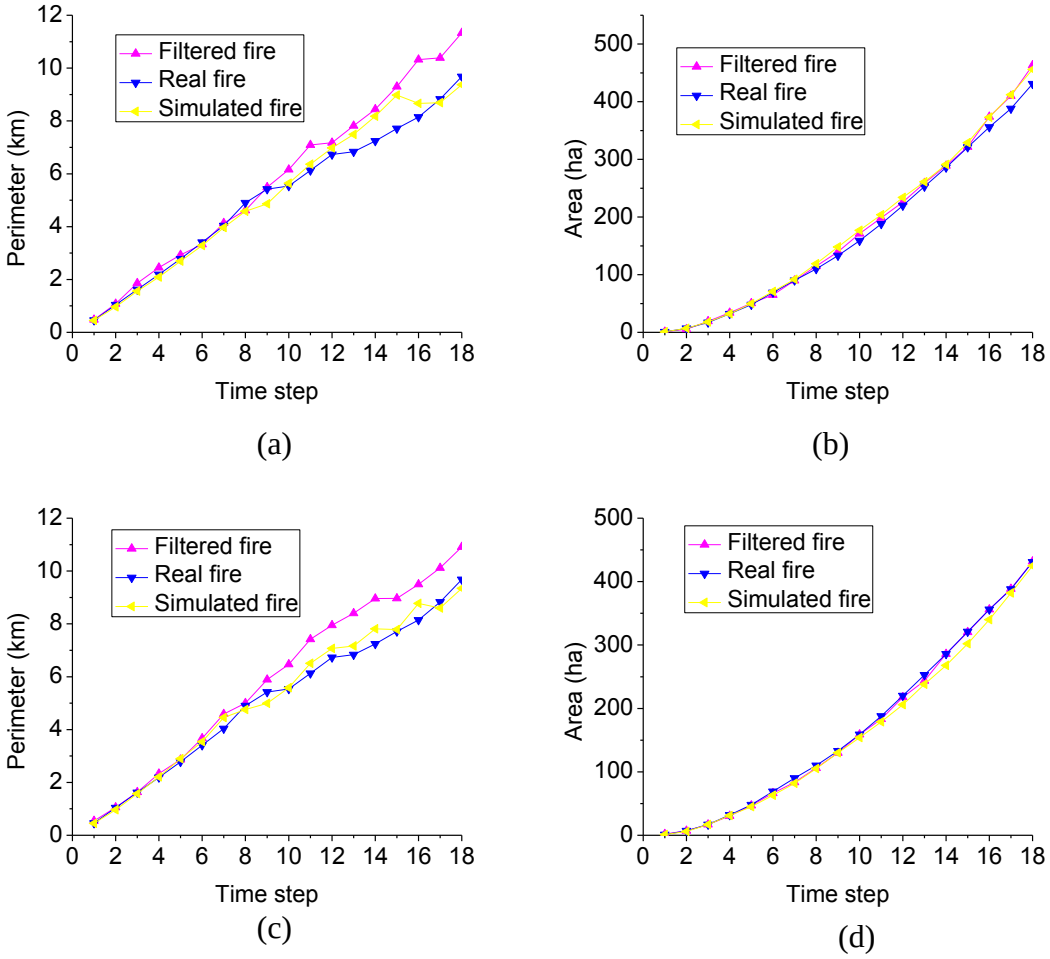


Figure 6.7 Perimeters and areas of real fire, simulated fires, and filtered fires for case 3 and case 4. (a) Perimeters for case 3; (b) Areas for case 3; (c) Perimeters for case 4; (d) Areas for case 4.

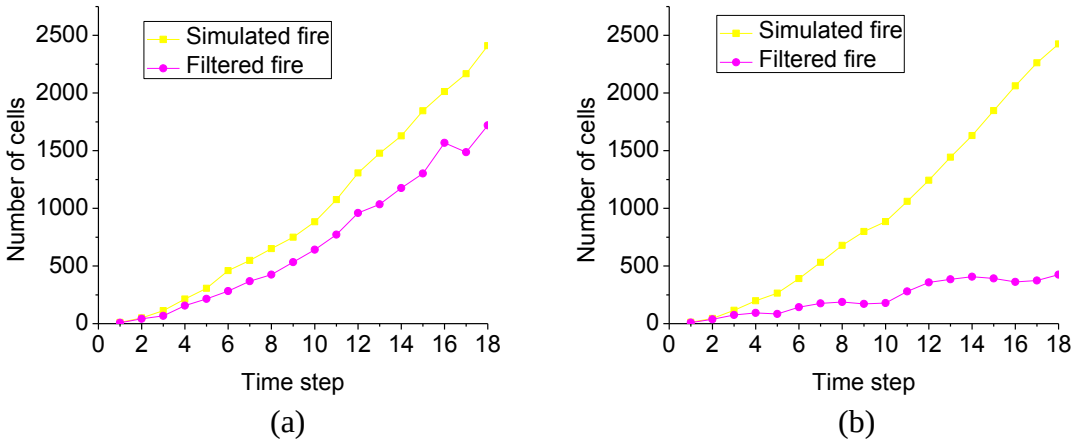


Figure 6.8 Symmetric set differences for case 3 and case 4. (a) Case 3; (b) Case 4.

CHAPTER 7 SENSITIVITY ANALYSIS

From Chapter 6, we know that by assimilating the observation data into DEVS-FIRE, the simulation results can be improved. But it is still needed to be explored how the factors influence the simulation results. Therefore, the sensitivity analysis will be done in the following subsections. The influences of errors between the real system and the simulation system, the sensor density, the frequency of sensor data, and the quality of sensor data on the simulation results will be examined.

7.1 Errors between the real system and the simulation system

To study how the errors affect the simulation results, we compare two sets of experiments as shown in Table 7.1. In the table, the real wind condition is assume to be 8 (miles/hour) and 180 degrees (from South to North), based on which the wind speed and the wind direction fluctuate -2 ~ 2 (miles/hour) and -20 ~ 20 (degrees) respectively every 10 minutes. For all the cases, the wind condition errors are larger than those in Chapter 6. For all the cases in Table 7.1, the sensors are regularly distributed every 4 cells.

Table 7.1 Experimental sets of different errors

<table>
<tr><th rowspan="2">Cases</th><th colspan="2">“Error” data</th><th colspan="2">Real data</th></tr>
<tr><th>Speed (miles/hour)</th><th>Direction (degrees)</th><th>Speed (miles/hour)</th><th>Direction (degrees)</th></tr>
<tr><td>5</td><td>4±2</td><td rowspan="2">No errors</td><td rowspan="4">8±2</td><td rowspan="4">180±20</td></tr>
<tr><td>6</td><td>12±2</td></tr>
<tr><td>7</td><td rowspan="2">No errors</td><td>140±30</td></tr>
<tr><td>8</td><td>220±30</td></tr>
</table>

We implemented data assimilation in DEVS-FIRE with imprecise data as shown in Table 7.1. The fire spreads (run DEVS-FIRE) with the real data and the error data are displayed in Figure 6.1(a) and Figure 7.1 (the burning cells and the burned cells are displayed in red and black respectively) respectively. Figure 7.1(a), Figure 7.1(b), Figure

7.1(c), and Figure 7.1(d) show the simulated fires for case 5, case 6, case 7, and case 8 respectively.

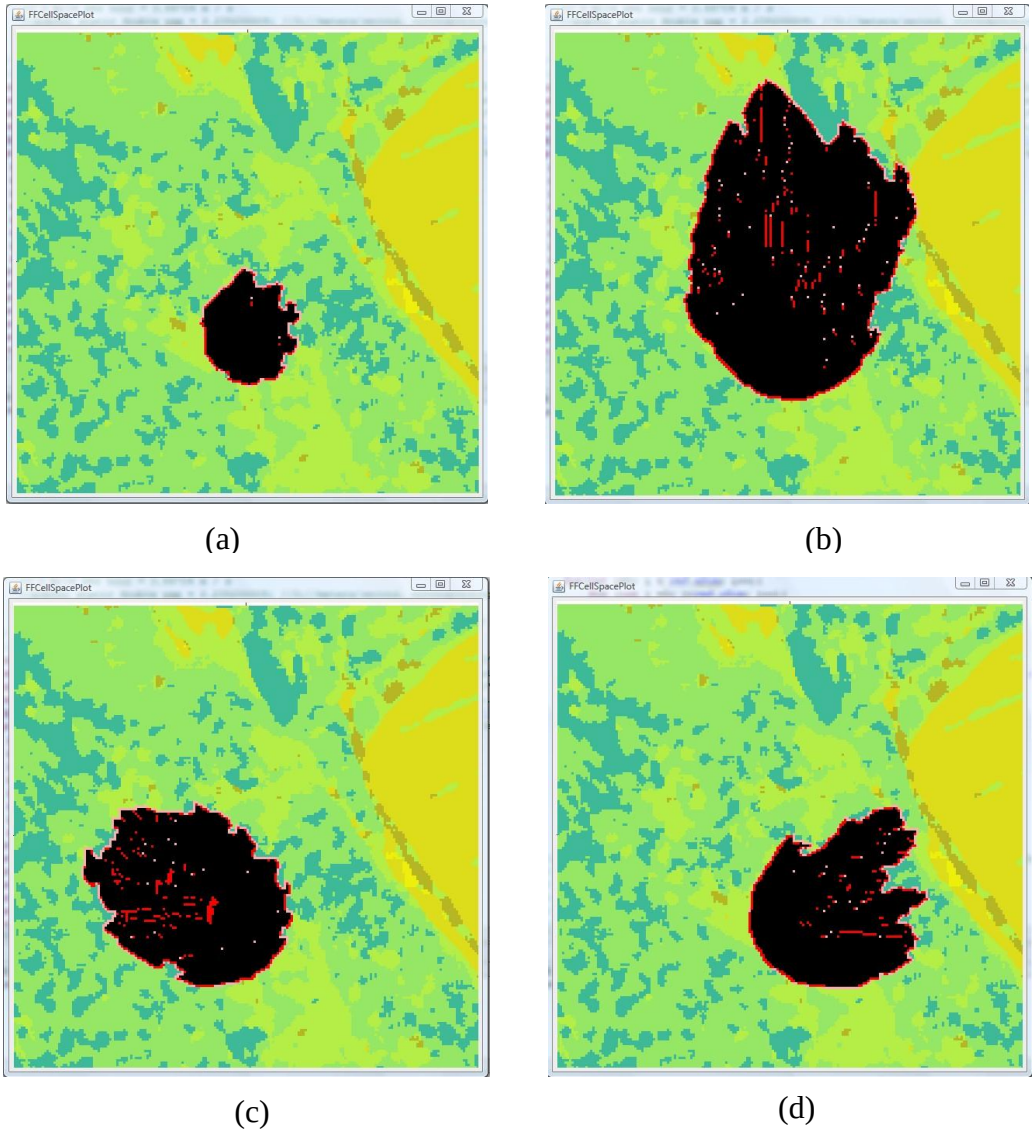


Figure 7.1 Simulated fires for case 5~8. (a) Simulated fire for case 5 after 360 minutes; (b) Simulated fire for case 6 after 360 minutes; (c) Simulated fire for case 7 after 360 minutes; (d) Simulated fire for case 8 after 360 minutes.

By assimilating the observation data into DEVS-FIRE, the filtered fires are obtained as shown in Figure 7.2. Figure 7.2(a), Figure 7.2(b), Figure 7.2(c), and Figure 7.2(d) display the filtered fires (the fire fronts are displayed in blue) compared with the real fire

(the fire front is displayed in red) and the simulated fires (the fire fronts are displayed in green) for case 5, case 6, case 7, and case 8 respectively.

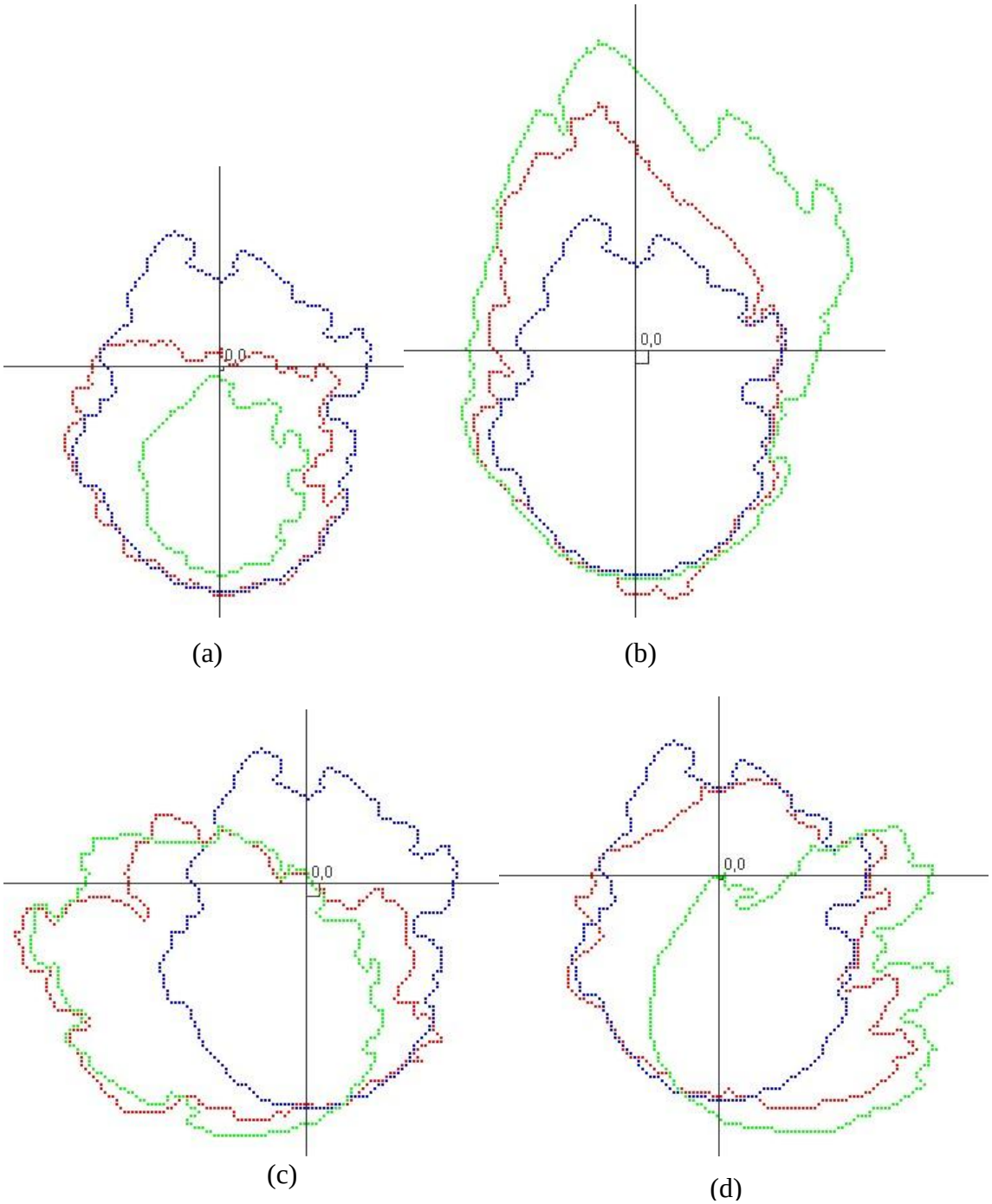
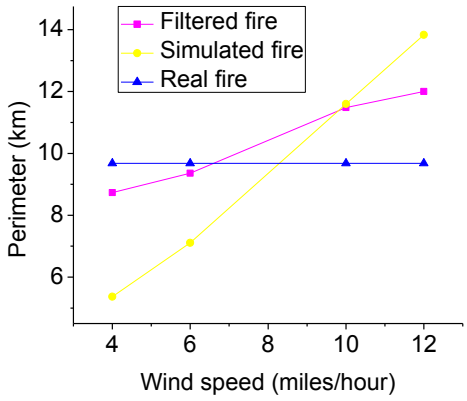
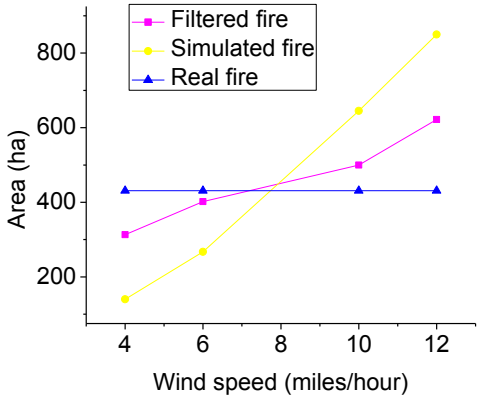


Figure 7.2 Comparisons of real fire, simulated fires, and filtered fires for case 5 ~ 8. (a) Case 5; (b) Case 6; (c) Case 7; (d) Case 8.

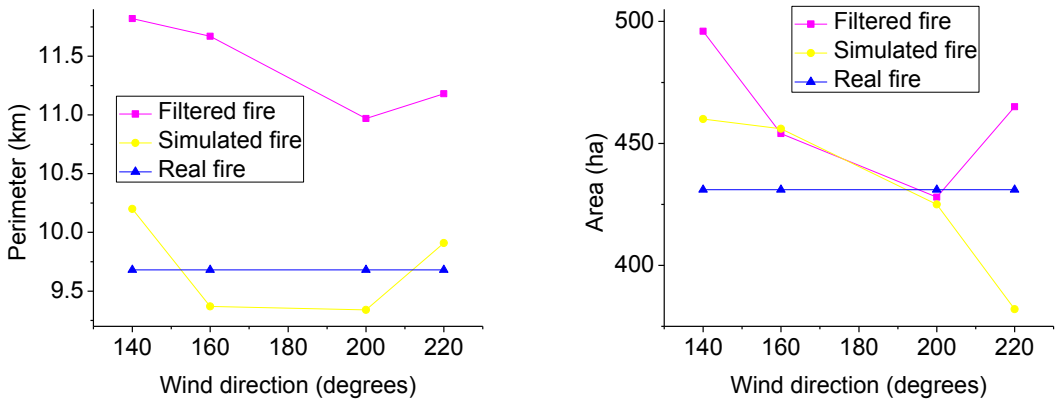
By comparing all the cases of Figure 7.2 with corresponding cases of Figure 6.2, we can see that the greater the errors between the real system and the simulation system, the worse simulation results we obtain if all other conditions are same. Figure 7.3 shows the relationships between wind conditions and the simulation results. In the figures, the horizontal axis represents the wind speed or wind direction, and the vertical axis refers to the burning areas/perimeters of the real fire, the simulated fires, and the filtered fires after 360 minutes. From Figure 7.3(a) and Figure 7.3(b), we know that the areas and perimeters of simulated fires become larger with the increase of the wind speeds, and closer to the real wind speed (8 miles/hour) the error wind speed, the better results could be obtained. From Figure 7.3(c) and Figure 7.3(d), we can see that closer to the real wind direction (180 degrees) the error wind direction, the better results could be achieved too.



(a)



(b)



(c) (d)

Figure 7.3 Relationship between the wind conditions and the perimeters and areas of real fire, simulated fires, and filtered fires. (a) Perimeter and wind speed; (b) Area and wind speed; (c) Perimeter and wind direction; (d) Area and wind direction.

Figure 7.4 shows the relationship between the wind conditions and the symmetric set differences between the simulated/filtered fires and the real fire after 360 minutes. From Figure 7.4(a) and Figure 7.4(b), we also see that for all the cases, the symmetric set differences between the filtered fires and the real fire are smaller than those between the real fire and corresponding simulated fires. Furthermore, the smaller difference between the real wind speed (direction) and the error wind speed (direction) is, more symmetric set differences are decreased. This also shows the similar conclusion as stated above.

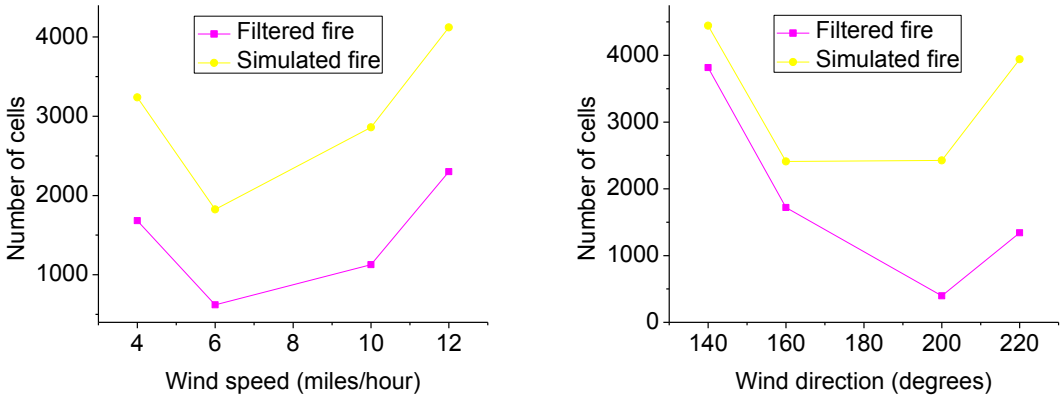
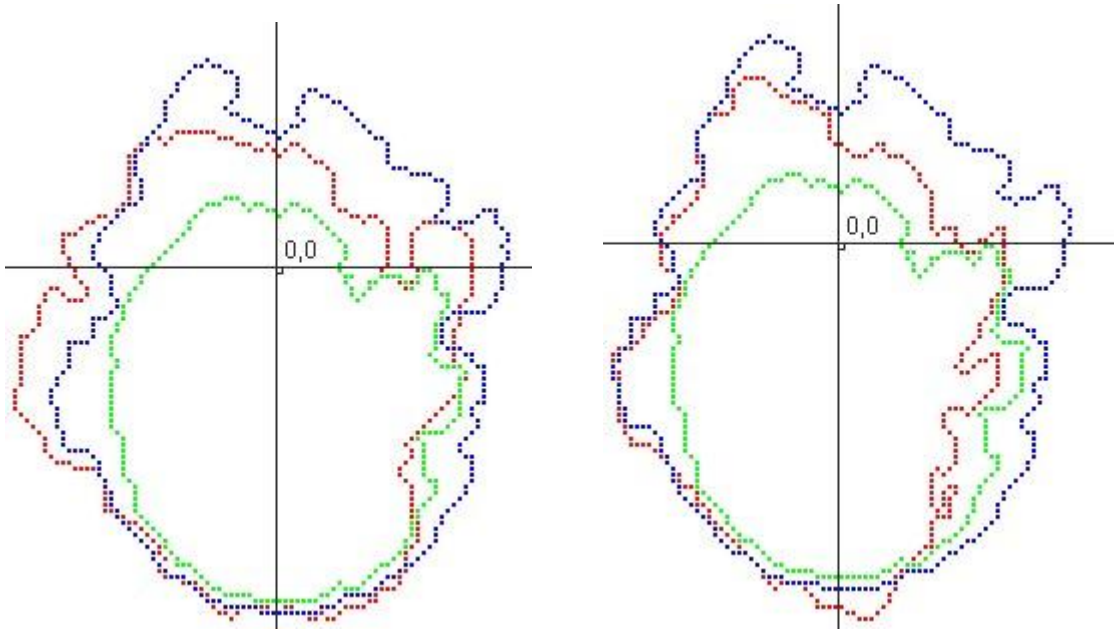


Figure 7.4 Relationship between symmetric set differences and wind conditions. (a) Symmetric set differences between the real fire and filtered/simulated fires with wind speed; (b) Symmetric set differences between the real fire and filtered/simulated fires with wind

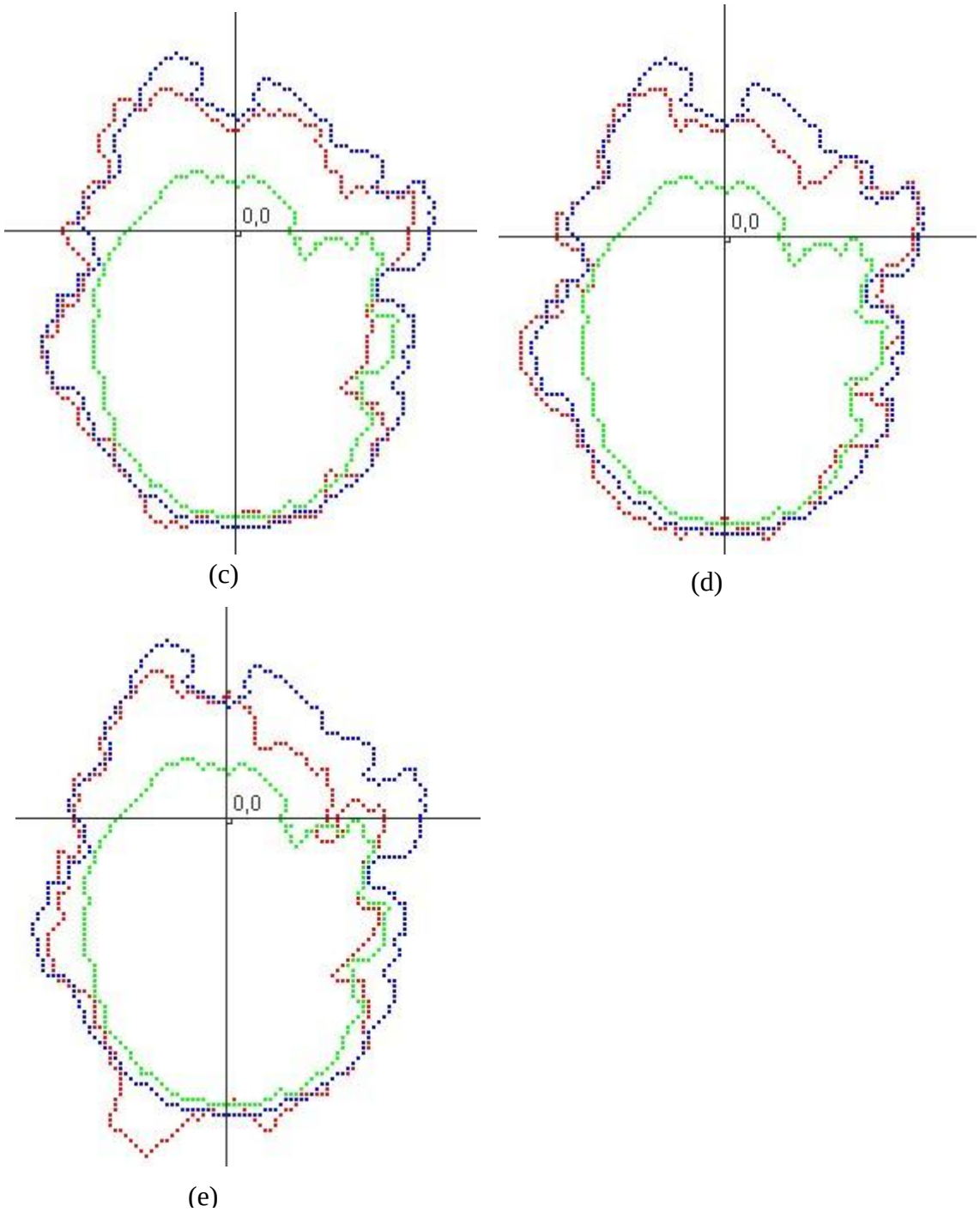
7.2 Sensor density

Sensor density is an important factor for sensor deployment, which is the number of sensors per unit. To examine its effects on simulation results, based on case 1 in Table 6.1, five various sensor deployment schemas (five different sensor densities) are used for data assimilation in DEVS-FIRE, including 400 sensors randomly deployed (density of 1/100 per cell), 625 sensors randomly deployed (density of 1/64 per cell), 1089 sensors randomly deployed (density of 1/36 per cell), 2500 sensors randomly deployed (density of 1/16 per cell), and 10000 sensors randomly deployed (density of 1/4 per 4 cell). The filtered fires (the fire fronts are displayed in blue) for these five sensor deployment schemas (densities) are displayed in Figure 7.5(a), Figure 7.5(b), Figure 7.5(c), Figure 7.5(d), and Figure 7.5(e), compared with their corresponding real fire (the fire front is displayed in red) and simulated fires (the fire fronts are displayed in green). From Figure 7.5(a), Figure 7.5(b), Figure 7.5(c), and Figure 7.5(d), we can see that with increase of the density of deployed sensor from 1/100 per cell to 1/16 per cell, the accuracy of the

prediction are also improved. However, when the density increases to very dense (1/4 per cell), the prediction accuracy is not obviously improved as shown in Figure 7.5(e). This is because the smallest distances from the burning cells to the sensors is large; therefore, all the sensors have relatively high temperatures. By this way, the real time data and the measurement data has little differences, thus to result in low capability to differentiate fire fronts.



(a) (b)



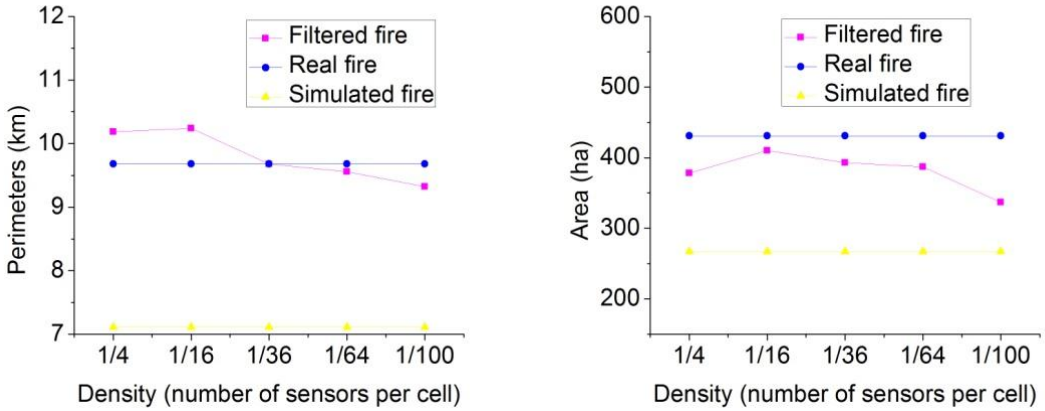
(c) (d)

(e)

Figure 7.5 Fire spreads with various sensor deployment schemas. (a) 1/100 per cell; (b) 1/64 per cell; (c) 1/36 per cell; (d) 1/16 per cell; (e) 1/4 per cell.

Figure 7.6 displays the relationship between the perimeters and areas of the real fire, the simulated fire, and the filtered fire at time step 18 and sensor densities. From the figures we see that, from the density of 1/16 per cell, perimeters of the filtered fires are closer to the real fire than the simulated fire, and areas of the filtered fires have more differences with the increase of sensor densities. Figure 7.7 shows the relationship between the decreased symmetric set difference (SSD) and the sensor densities. The decreased SSD defines how much SSD is decreased by using data assimilation as shown in equation (7.1). In the equation, $SSD_{filtered}$ and $SSD_{simulated}$ refer to the SSD between the filtered fire and the real fire, and that between the simulated fire and the real fire respectively. The bigger the decreased SSD, the more the simulation results are improved. Also we know that from density of 1/16 per cell, the decreased SSD becomes smaller with the decrease of the sensor densities. This is incidence with Figure 7.6(b).

$$\text{decreased SSD} = (SSD_{filtered} - SSD_{simulated})/SSD_{simulated} \quad (7.1)$$



(a) (b)

Figure 7.6 Fire perimeters and areas of real fire, simulated fire, and filtered fires with sensor densities. (a) Perimeters with sensor densities; (b) Areas with sensor densities.

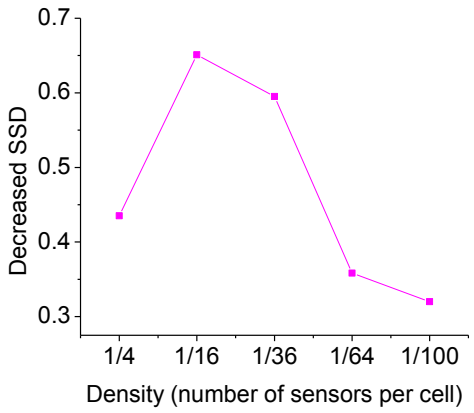
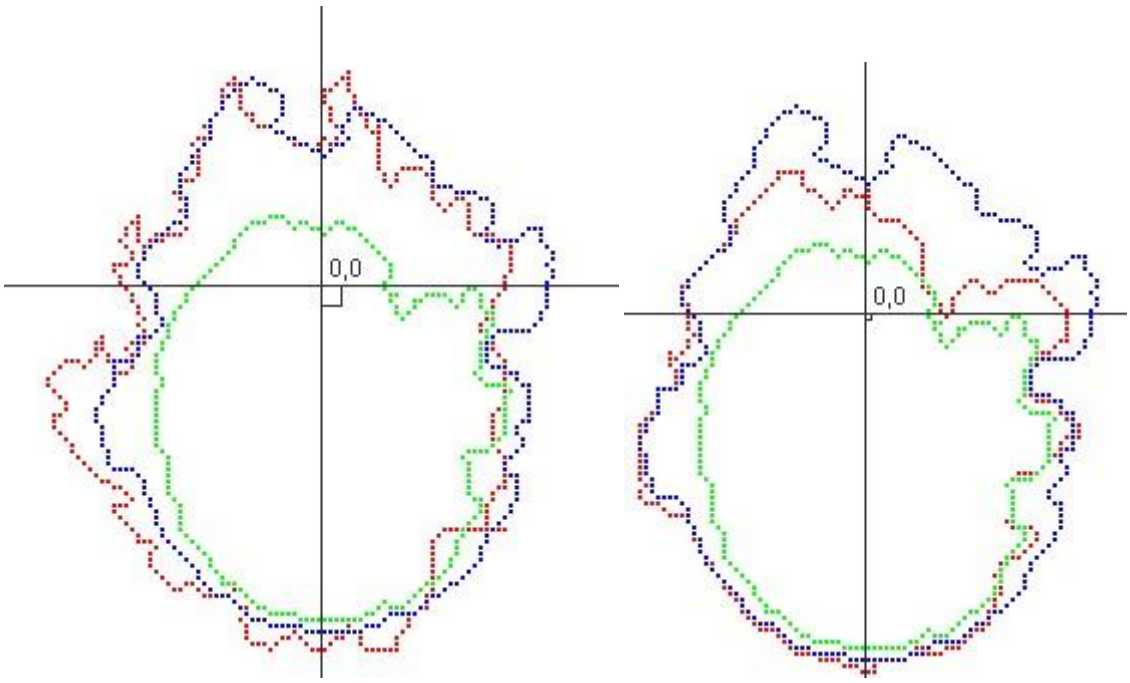


Figure 7.7 Decreased symmetric set differences (SSD) with sensor densities

7.3 Frequency of sensor data

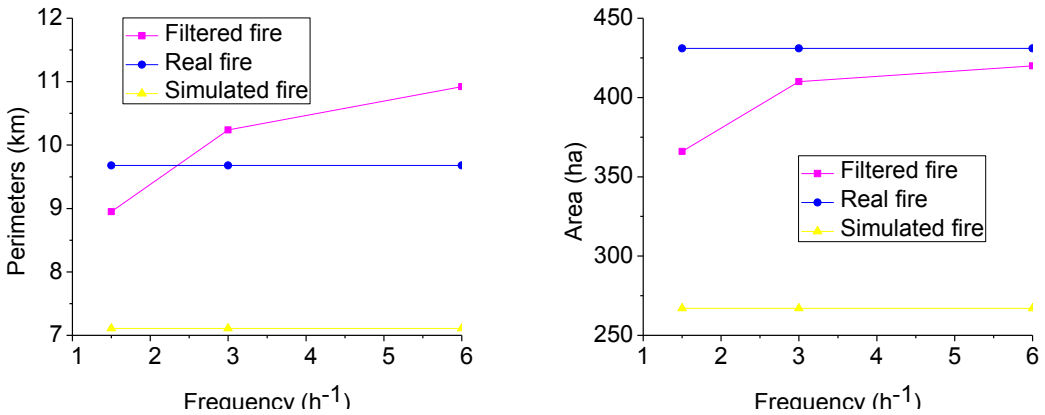
Frequency of sensor data refers to the number of times to assimilate real time data per hour. To examine the influences of the sensor data's frequency on simulation results, we also implement case 1 in Table 6.1 using three difference frequencies (6 per hour, 3 per hour, and 1.5 per hour). Figure 7.8 displays the real fire (display in blue), the simulated fire (display in green), and the filtered fires (display in red) for the frequencies of 6 per hour and every 1.5 per hour compared with those for frequency of 3 per hour as shown in Figure 7.5(d). By comparing Figure 7.8(a), Figure 7.8(b), and Figure 7.5(d), we can see that the data frequency affects the simulation results. For a high frequency, the simulation result has more real time information to be utilized, thus the filtered fire has more details as shown in Figure 7.8(a). With the data frequency decreases to 1.5 per hour, the simulation result is obviously worse as shown in Figure 7.8(b).



(a) (b)

Figure 7.8 Fire spreads with various data frequencies. (a) 6 per hour; (b) 1.5 per hour.

Figure 7.9 shows the perimeters and areas of the real fire, the simulated fire, and the filtered fires for three different data frequencies. From the figures we know that the simulation with frequency of 1.5 per hour is obviously worse than ones with frequencies of 6 per hour and 3 per hour. The simulation with the frequency of 6 times per hour has a little worse result than that with the frequency of 3 per hour. This is because the former has more detailed noise generated on the fire fronts, thus to generate more noises than the propagation of the time step. Figure 7.10 displays the relationship between the decreased symmetric set difference (SSD) and the sensor data frequencies. Also we know that from frequency of 3 per hour, the decreased SSD becomes smaller with the decrease of the sensor data frequencies. This is incidence with Figure 7.8(b) except the very frequent case of 6 times per hour.



(a) (b)

Figure 7.9 Perimeters and areas of real fire, simulated fire, and filtered fires with various data frequencies. (a) Perimeters; (b) Areas.

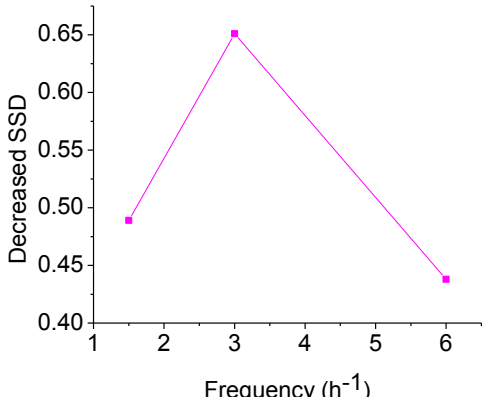


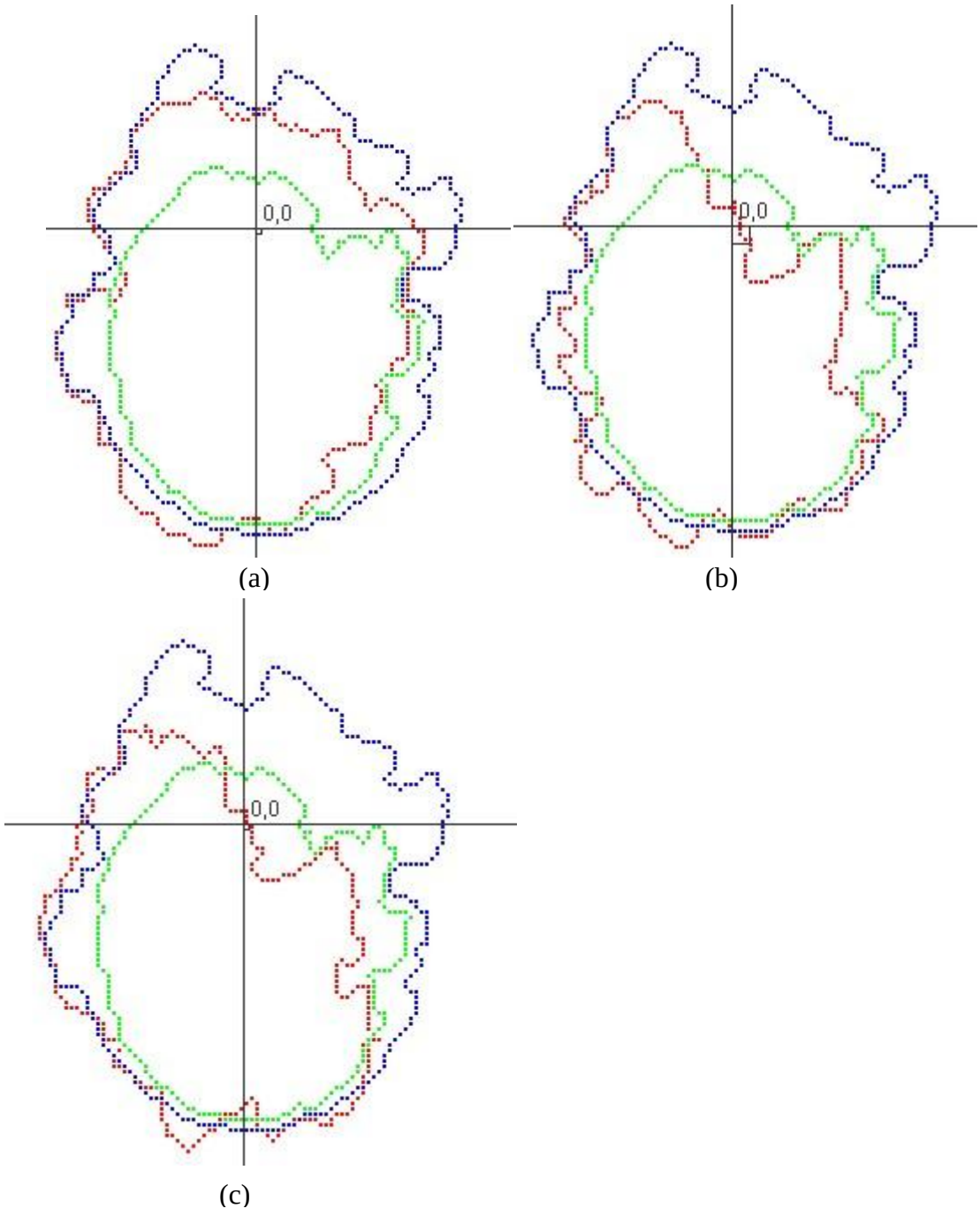
Figure 7.10 Decreased symmetric set differences (SSD) with sensor data frequencies

7.4 Quality of sensor data

The next factor will be explored is the quality of sensor data. For this goal, we manually add some errors to the sensor data, including 10%, 20%, and 40%. Based on the different error observation data, we ran the DDDAS and obtained related results. Figure 7.11 shows the real fire (display in blue), the simulated fire (display in green), and corresponding filtered fires (display in red) for three different degrees of errors. From the

figures, we know for the case with 10% sensor data error, the simulation result was improved by running DDDAS. But, for two cases of 20% and 40% sensor data error, the simulation results are worse than the simulated fire. Therefore, the quality of the observation sensor data is a very important factor to affect the accuracy of simulation results.

Figure 7.12 shows the perimeters and areas of the real fire, the simulated fire, and corresponding filtered fires for four different cases, which have the sensor data errors of 0%, 10%, 20%, and 40% at time step 18. From the figures, we know that the areas of the filtered fires are closer to the real fire with the decrease of the sensor data errors. But this is not the case for perimeters because the perimeters cannot completely show the features of fire fronts. For example, although the area of filtered fire is much smaller than the real fire, their perimeters are very close as shown in Figure 7.11(b). Figure 7.13 displays the decreased symmetric set difference for four cases with different sensor data above. According to the figure, we can see that the simulation results become worse and worse with the increase of sensor data error. Specifically, when the error increases to 40%, the symmetric set difference between the real fire and the filtered fire is larger than that between the real fire and the simulated fire. It has the same trend as shown in Figure 7.10(b).



(a) (b)

(c)

Figure 7.11 Fire spreads with various sensor data errors. (a) 10%; (b) 20%; (c) 40%.

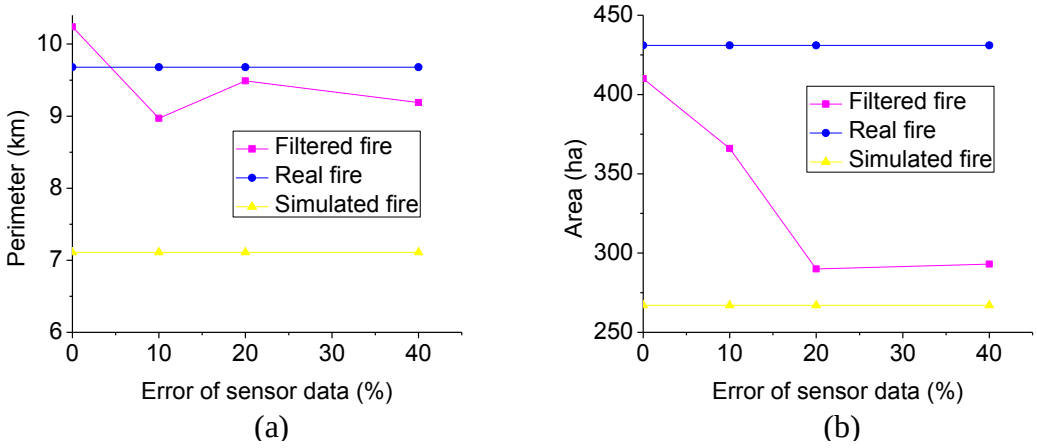


Figure 7.12 Perimeters and areas of real fire, simulated fire, and filtered fires with different sensor data errors. (a) Perimeters; (b) Areas.

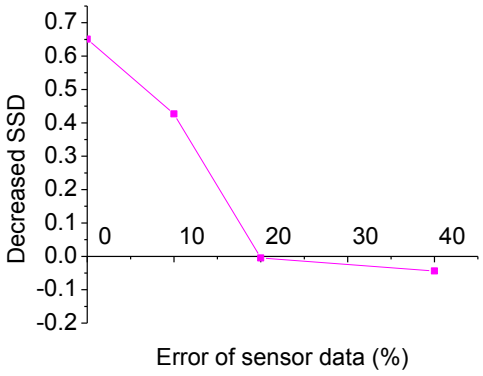


Figure 7.13 Decreased symmetric set difference (SSD) with sensor data errors

CHAPTER 8 MEASUREMENT ANALYSIS

8.1 Measurement metrics

To analyze the robustness of DDDAS based on SMC methods, we use several metrics that measure the effectiveness of the developed method from different aspects. These include convergence, degeneracy, and sample impoverishment. These metrics are general for any application. Specific details (equations) for DDDAS for wildfire fire spread application are presented later.

8.1.1 Convergence metrics

When applying SMC methods in applications, an important question to be considered is convergence. In many statistics literatures, this question has been theoretically answered, such as (Crisan, Del Moral, and Lyons 1999; Crisan 2001). To make the proofs understandable, (Crisan and Doucet 2002) presented the convergence results of SMC methods in the simple way. In spite of these promising results of SISR and its improved versions, it is still difficult for practitioners to use because all of them explore the situation where the number of particles approaches infinity. In real applications, we only have limited computing resources. Therefore, the measures to calibrate the convergence results are needed. Below we will discuss two of related approaches.

To evaluate how confidently the ensemble can be obtained from its spread, the normalized measure of ensemble dispersion can be used. According to the method discussed in (Anderson 2001), *NRR*, the normalized Root Mean Square Error (RMSE), is defined as $R_1 / R_2 / \sqrt{(n+1)/2n}$, where n is the ensemble size; R_1 is the time-averaged RMSE of the ensemble mean as shown in equation (8.1); R_2 is the mean RMSE of the

ensemble members as shown in equation (8.2). In the equations, *T* is the analysis time period. Using *NRR*, we can evaluate the error propagation. If *NRR* is larger than 1, it means the ensemble has too little spread. *NRR* < 1 indicates the ensemble has too much spread. The expected case is that the value of *NRR* is 1. More details about this measure and its explanation can be found in (Anderson 2001).

$$R_1 = \frac{1}{T}\sum_{t=1}^{T}\sqrt{\left[\left(\frac{1}{n}\sum_{i=1}^{n}\hat{y}_t^i\right) - y_t^i\right]^2} \ . \tag{8.1}$$

$$R_2 = \frac{1}{n}\sum_{i=1}^{n}\sqrt{\frac{1}{T}\sum_{t=1}^{T}(\hat{y}_t^i - y_t^i)^2} \ . \tag{8.2}$$

Another measure of *ER*, short for Exceedence Ratio, is used to evaluate the spread of prediction quantiles (Borga 2002). Its definition is shown in equation (8.3), where ER_n and N_{exc}^n means the exceedence ratio at *n*th percentile and the number of times during the total number of analysis period of *T*. If the uncertainty range to generate the samples is too wide, the model has high predicative capability and low precision, and vice versa.

$$ER_n = \frac{N_{exc}^n}{T} \times 100\% . \tag{8.3}$$

8.1.2 Degeneracy metric

Degeneracy refers to the phenomenon that the samples gradually diverge. If that happens, the estimated states are far away from the true states and we cannot obtain "correct" results. For the SIS (SMC methods without resampling), if the variance of w_t stochastically increases over time, only one sample will have all the weight, leading to degeneracy problem (Doucet, Gordon, and Krishnamurthy 1999). Figure 8.1 shows the scenario with degeneracy at time step 1, 2, and 3 from left to right. In the figures, the horizontal axis and the vertical axis represent the samples and their weights respectively. From the figures, we know that the distributions have larger differences over time

because the variance of w_t becomes larger and larger. To measure this kind of degeneracy, a measure of effective sample size (N_{eff}) was introduced in (Liu 1996) as shown in equation (8.4). From the equation, we know that if only one sample dominants, N_{eff} has the value of 1 indicating a severe degeneracy. Therefore, to reduce degeneracy, we try to increase N_{eff}; for example, the uniformly distributed samples will make it the number of samples N.

$$N_{eff} = \frac{1}{\sum_{i=1}^{N} (w_t^i)^2} \quad . \tag{8.4}$$

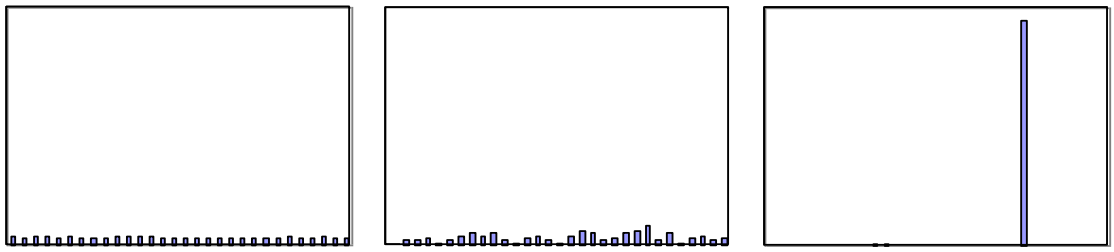


Figure 8.1 Degeneracy illustration

To solve degeneracy above, a resampling stage is introduced. In the step, we try to remove the samples with low weights and generate multiple copies of sample with high weights. The resampling step maps the weighted samples $\left\{x_{0:t}^i, \tilde{w}_t(x_{0:t})\right\}$ to samples with equal weights $\left\{x_{0:t}^j, N^{-1}\right\}$.

8.1.3 Sample impoverishment

Sample impoverishment occurs when most of the samples are fallen into a small region of the solution space. This is not what we expect. Note that SISR may cause sample impoverishment because a particle with high importance weight could duplicate multiple times, thus to introduce high correlations between particles. To reduce sample

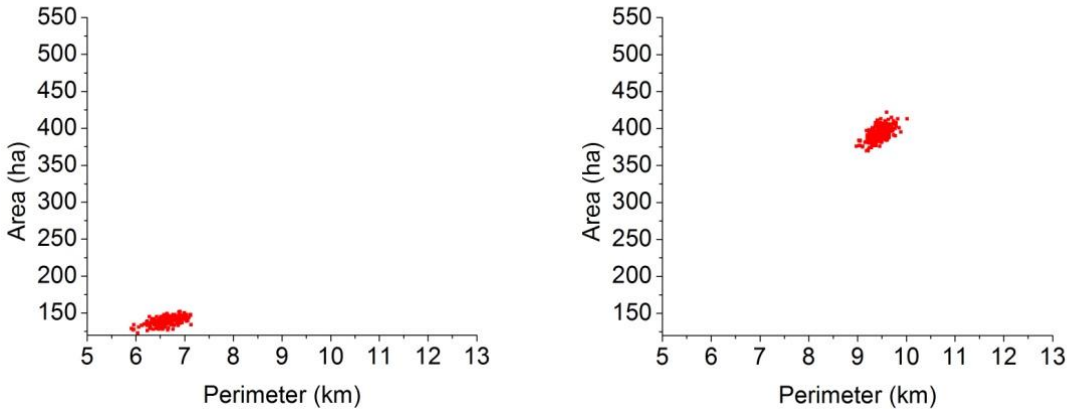
impoverishment, on one hand, we can increase the number of samples. This will make the samples have more “diversity”, but computational complexity will be increased. On the other hand, we can achieve this goal by enlarging the noises of the system state variables. To detect the sample impoverishment, the range of samples from the minimum to maximum can be used to measure their diversity.

8.2 Analysis of SMC methods in DEVS-FIRE

To evaluate simulation results of SMC methods in DEVS-FIRE, all above measurement metrics should be used. Therefore, two sets of experiment (case 1 in Chapter 6 and case 3 in Chapter 7) were chosen to be measured by the related metrics.

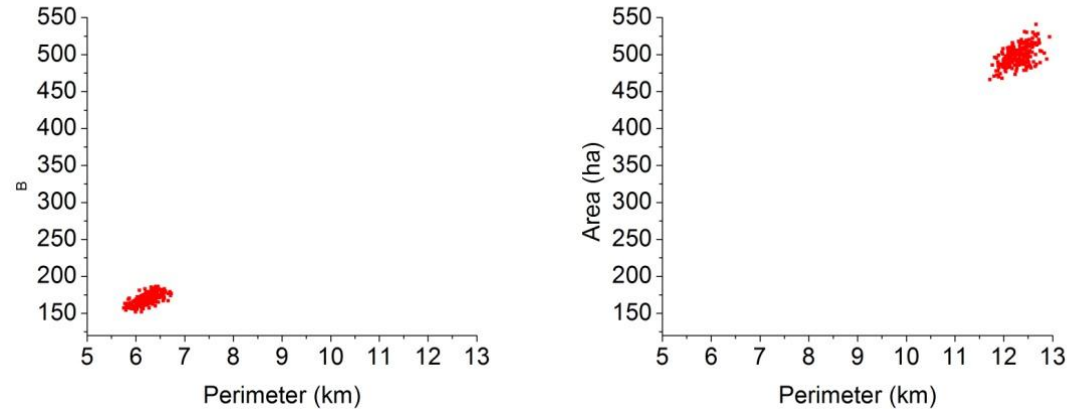
8.2.1 Convergence analysis

To examine the convergence, we compare distributions of particles for two cases. We use perimeter and area of each particle (fire front) for simplicity. Figure 8.2 shows the distributions of particles for the first case at time step 10 and 18 respectively. Figure 8.3 shows the distributions of particles for the second case at time step 10 and 18 respectively. In the figures, the horizontal axis and the vertical axis are the fire perimeters and the burning areas of the predicted fires. From the figures we know that with time advancing (from time step 10 to 18), the particles converge to small areas close to the real values (the fire perimeter and the area are 9.8 km and 431 ha) for both of cases. Moreover, the first case has a better simulation result because more particles are distributed around the real values. Note that the fire perimeter and the burning area cannot equally represent a fire front; therefore, they only partially reflect the convergence results.



(a) (b)

Figure 8.2 Convergence of the first case. (a) Time step 10; (b) Time step 18.

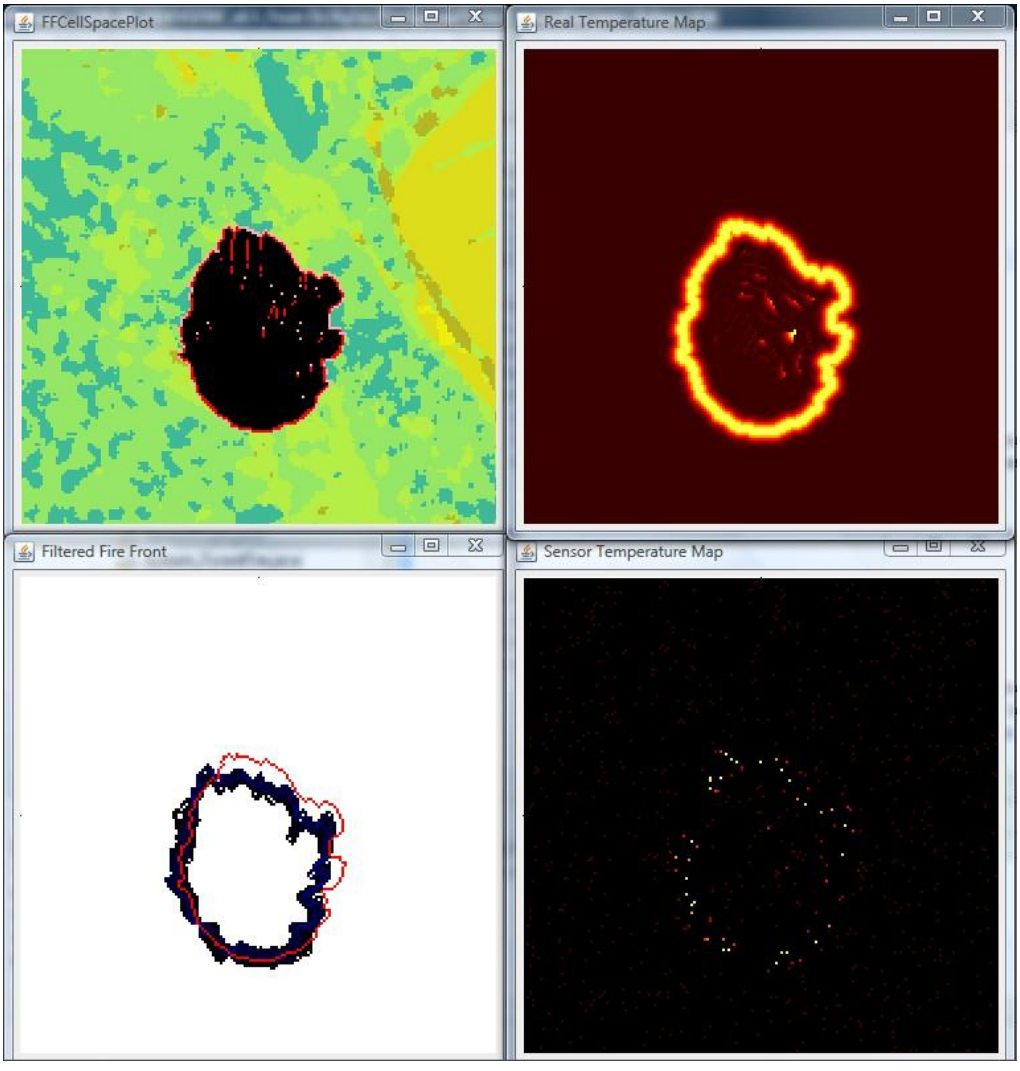


(a) (b)

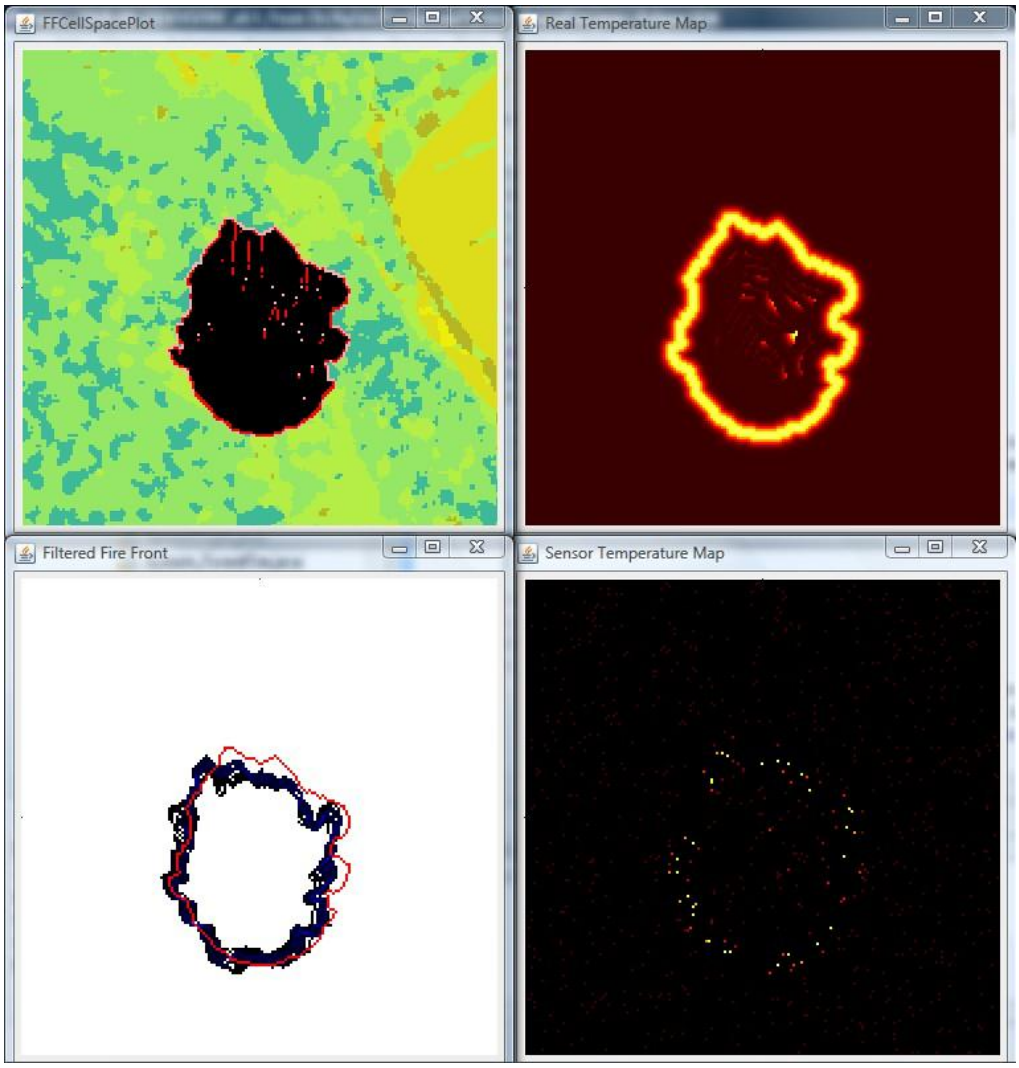
Figure 8.3 Convergence of the second case. (a) Time step 10; (b) Time step 18.

The whole procedure of SMC methods by assimilating sensor data into DEVS-FIRE (time step 15 to time step 18) is displayed in Figure 8.4. In the figures, the left top window is the real wildfire spread, the right top is the real temperature map, the left bottom window is the fire spread with DDDAS, and the right bottom window is the temperature map. From the description above, we know that totally we randomly distribute 1089 sensors in the cell space. In the window of fire spread with DDDAS, the

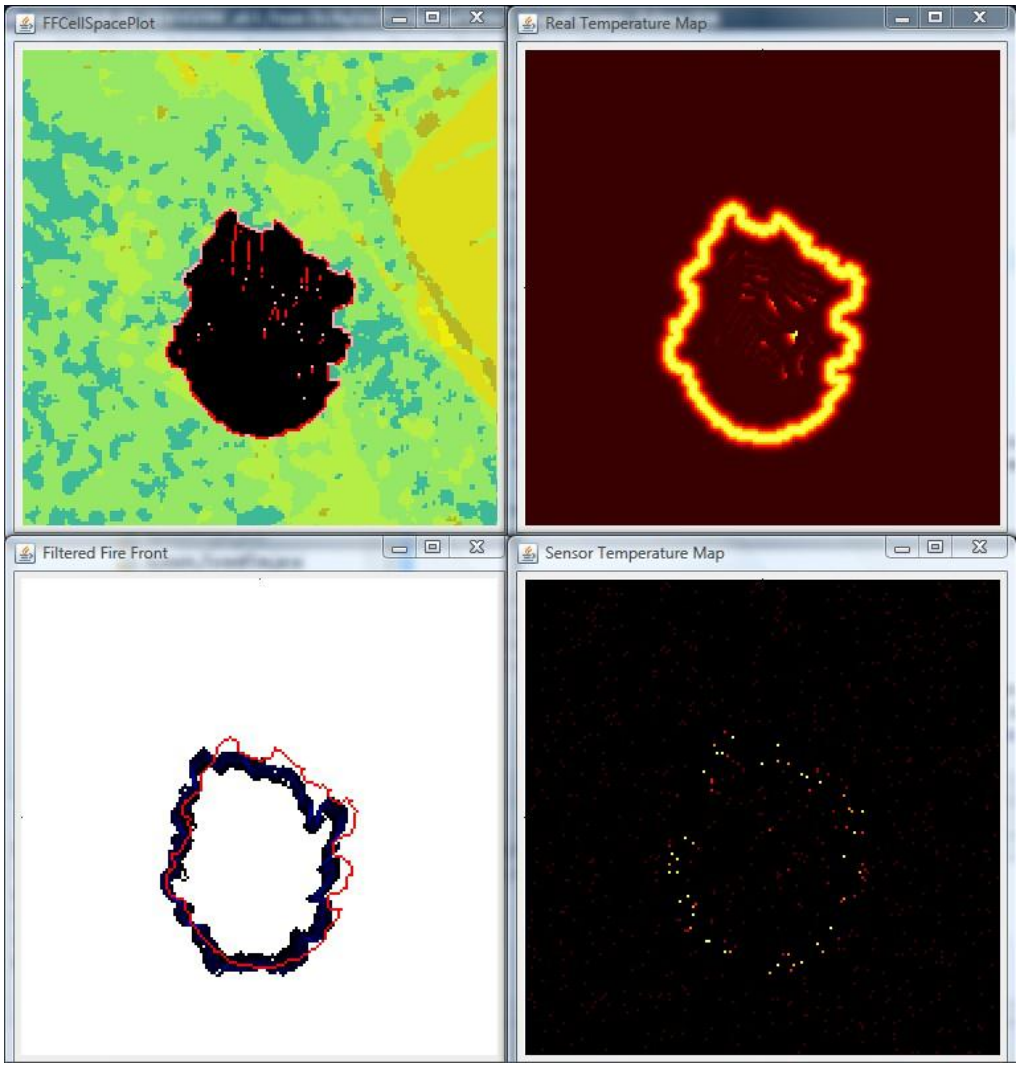
red cells represent the real fire, and the blue cells are the burned cells whose shades refer to their burning possibilities. The bigger the shade, the more possibly the cell is burned. From all the pictures, we know that each time step, the new sensor data is available and assimilated into the DEVS-FIRE model, thus to evaluate the fire fronts and generate new fire fronts for next step. In this case, the generated fire fronts with noises are close to the real fire front, and have relatively good convergence.



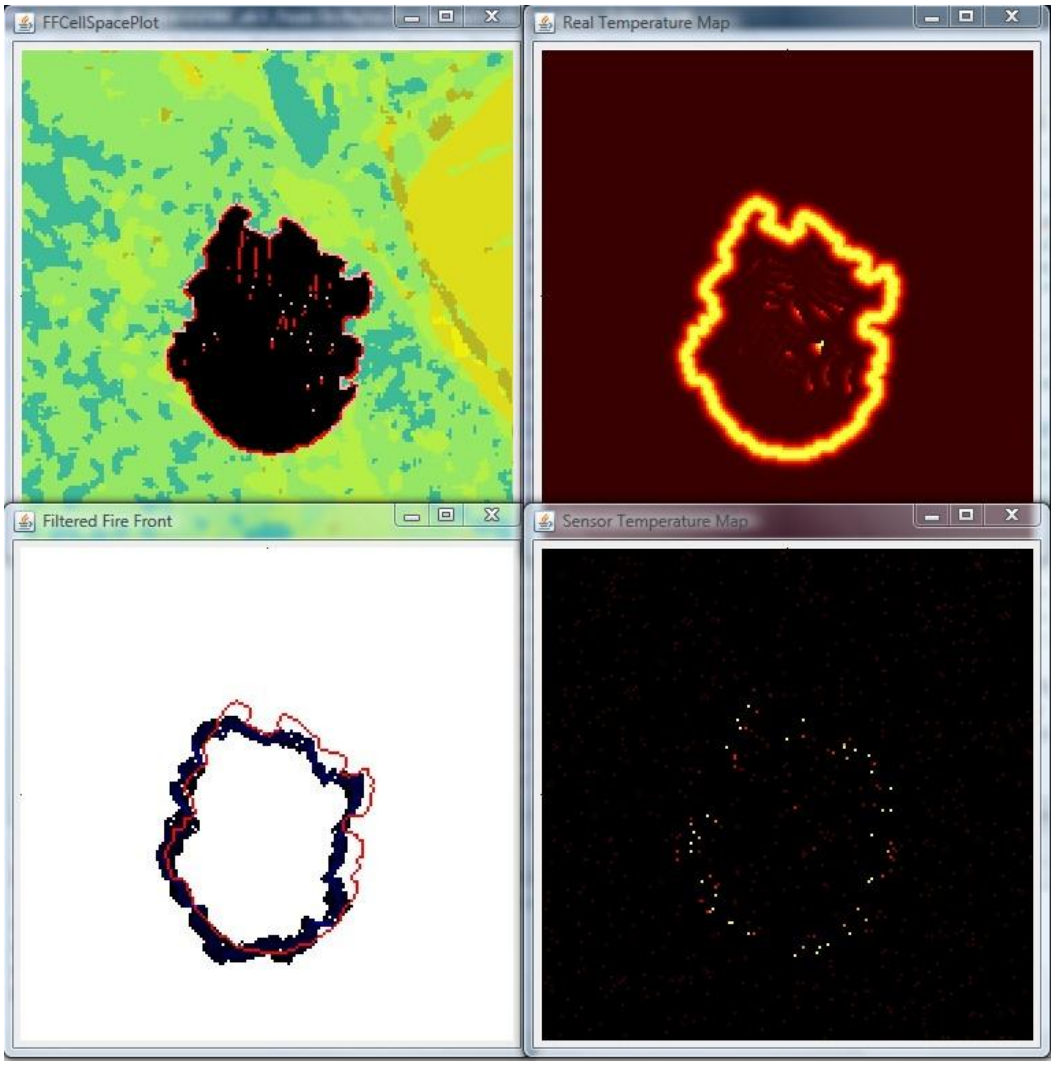
(a)


FFCellSpacePlot
Real Temperature Map
Filtered Fire Front
Sensor Temperature Map

(b)


FFCellSpacePlot
Real Temperature Map
Filtered Fire Front
Sensor Temperature Map

(c)



(d)

Figure 8.4 Convergence procedure of the first case. (a) Time step 15; (b) Time step 16; (c) Time step 17; (d) Time step 18.

In addition, the measures of *NRR* and ER_n are used to evaluate two cases above. Assume the analysis period is 5, which means the time step 14 to 18 is under consideration. For the first case, *NRR*1=1.1. For the second case, *NRR*2=1.4. Therefore, the ensemble of both cases has little spread, but the first case is with a very small value of uncertainty. This also verifies their simulation results. Regarding another measure of Exceedence Ratio, we compute them according to different ensemble percentile from 20

to 100 for both case 1 and case 2 as shown in Figure 8.5. From the figure, we see that *ER* sharply decreases with the increase of ensemble percentile. It explains the confidence level of the estimation.

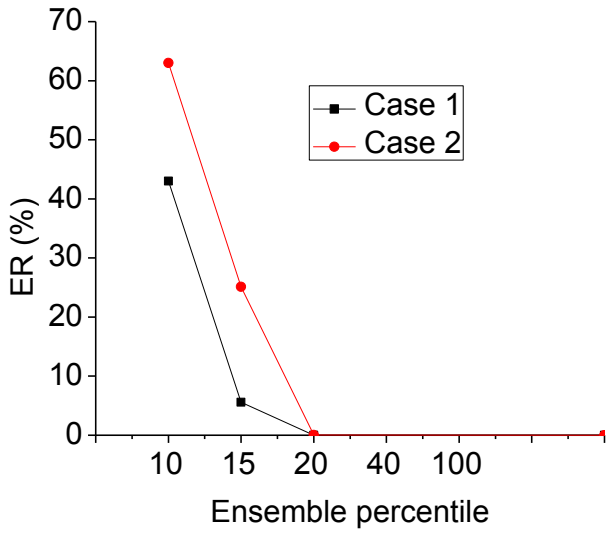


Figure 8.5 Exceedence ratios of case 1 and case 2

8.2.2 Degeneracy analysis

For two cases, we compute their effective sample sizes respectively. For case 1, its effective sample size $N_{eff\,1} = 2.25$. For case 2, its effective sample size $N_{eff\,2} = 1.11$. Therefore, the case 1 has more degeneracy than case 2. Because the effective sample size of case 2 is close to 1, its degeneracy is very severe. According to the degeneracy analysis, we can conclude that the resampling stage cannot completely remove degeneracy.

8.2.3 Sample impoverishment analysis

As stated above, the resampling step may lead to sample impoverishment. To evaluate the sample impoverishment, the fire perimeter and burning areas are also used. Figure 8.6 shows the ranges of area errors and perimeters errors of case 1 from time step

14 to 18. From the figures, we can see that during this period, sample impoverishment occurs because of ranges' decreasing with time elapsing. Especially, the range of area only lies below the 0. Figure 7.7 displays the ranges of area errors and perimeters errors of case 2 from time step 14 to 18. From the figures we know that case 2 shows the same trend to generate sample impoverishment. By comparing Figure 8.6 and Figure 8.7, we can conclude that case 2 has more severe sample impoverishment than case 1. More superficially, the solution space is small and above 0 in case 2, meaning that all generated particles are larger than the real fire. This results from the bigger imprecise wind direction. To solve this problem, the value of noises could be increased so that the solution space becomes larger, thus to generate diversity of the fire fronts.

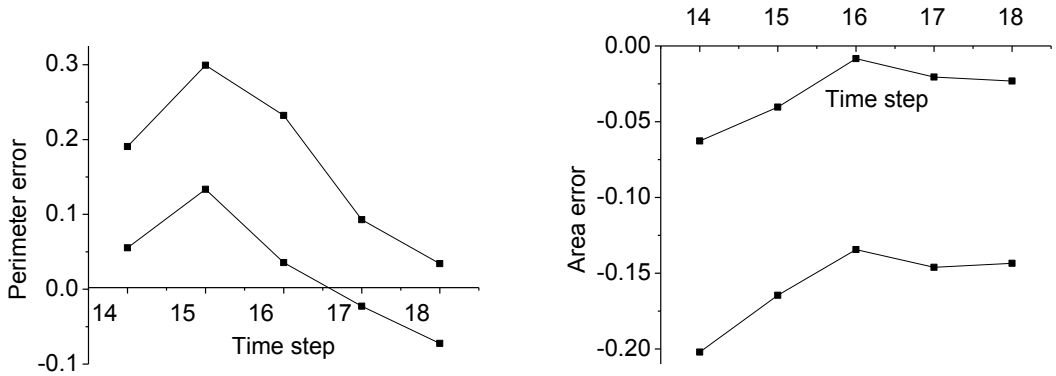


Figure 8.6 Sample impoverishment analysis for case 1

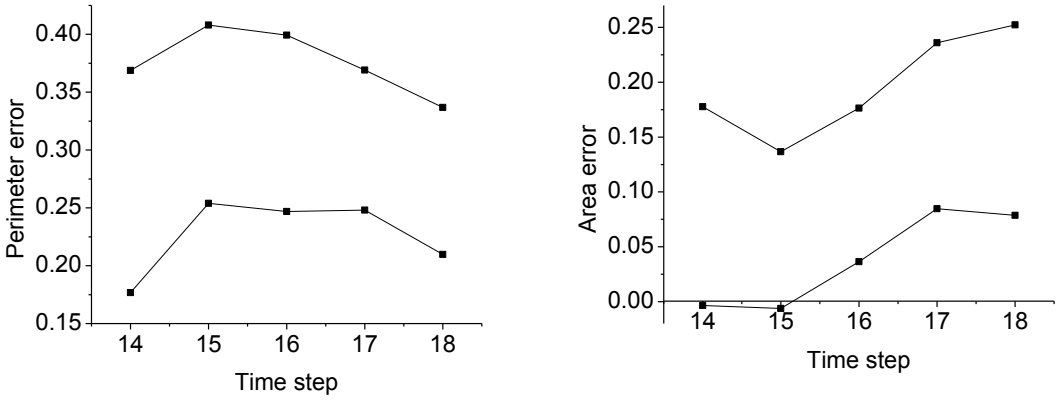


Figure 8.7 Sample impoverishment analysis for case 2

CHAPTER 9 APPLYING IMAGE MORPHING TO DDDAS

9.1 Basics of image morphing

Although we generate random noises to be added to the fire front, thus to model uncertainty in the state estimation, the possible fire fronts are still hard to be created because of the complexity of a shape and limited computational resources. We know that the fire front, which is a matrix to represent all the cells' states in the simulation, has the features of images containing a value for each pixel. In the SMC methods algorithm for DEVS-FIRE above, we decide the possibility of fire fronts by comparing the generated temperature map and the observation temperature map. Therefore, another way to utilize the observation data is to incorporate information from real system into the predicted states. This is what image morphing exactly does.

Image morphing is a technique combining two images into one through a smooth transition. Basically, two procedures are involved in image morphing, which are image warping and color interpolation. Image warping utilizes the 2 dimensional geometric transformations to align two images by their features, and color interpolation decides the values of each pixel of the warped image. Also, image morphing is widely used in many fields since the work of Smythe in 1988. With its receiving much attention, different kinds of algorithms in image morphing were proposed to solve related problems. In (Wolberg 1998), the author did a survey to talk about related issues in image morphing. Among these morphing algorithms, (Smythe 1990) and (Beier and Neely 1992) proposed the method of feature based morphing, using meshes to mark feature points and generate warping, and improve the method by applying line pairs to do warping respectively. Although it is still an active research area recently, the focus of image morphing is how

to automatically and effectively extract feature points and feature lines according to images based on the framework above.

9.2 Applying image morphing in DEVS-FIRE

Considering dynamic state estimation in DEVS-FIRE, we have two temperature maps according to the real time data and the predicted fire front. Therefore, we try to incorporate the features from both of these temperature maps to warp the predicted fire front. In this work, we extract multiple pair lines from two temperature maps, and warp the fire front to another by finding the new position of each cell using weighted combination of points defined by the line pairs as follows in equation (9.1).

$$w(i) = \left(\frac{l(i)^p}{a + d(i)}\right)^b, \tag{9.1}$$

where $w(i)$ is the weight of line i; $l(i)$ refers to the length of line i; $d(i)$ means the distance from the point to line i; a, b, p are constants to control the warping. More details about this algorithm can be found in (Beier and Neely 1992). Figure 9.1 shows the procedure to use image morphing to warp the fire front in DEVS-FIRE. Figure 9.1(a) and Figure 9.1(c) are the predicted temperature map and the observation temperature map respectively. According to extracted information from them, the fire front as shown in Figure 9.1(b) is warped to the shape as displayed in Figure 9.1(d).

From the figures above, we can see the morphed fire front is based on the predicted fire front, and transformed by incorporating the information from the observed data. Comparing the two temperature maps, obviously the observation has a larger width and smaller height than those of the prediction. From the results of image morphing, we also can notice this difference between the two fire fronts. This is because of applying morphing algorithm.

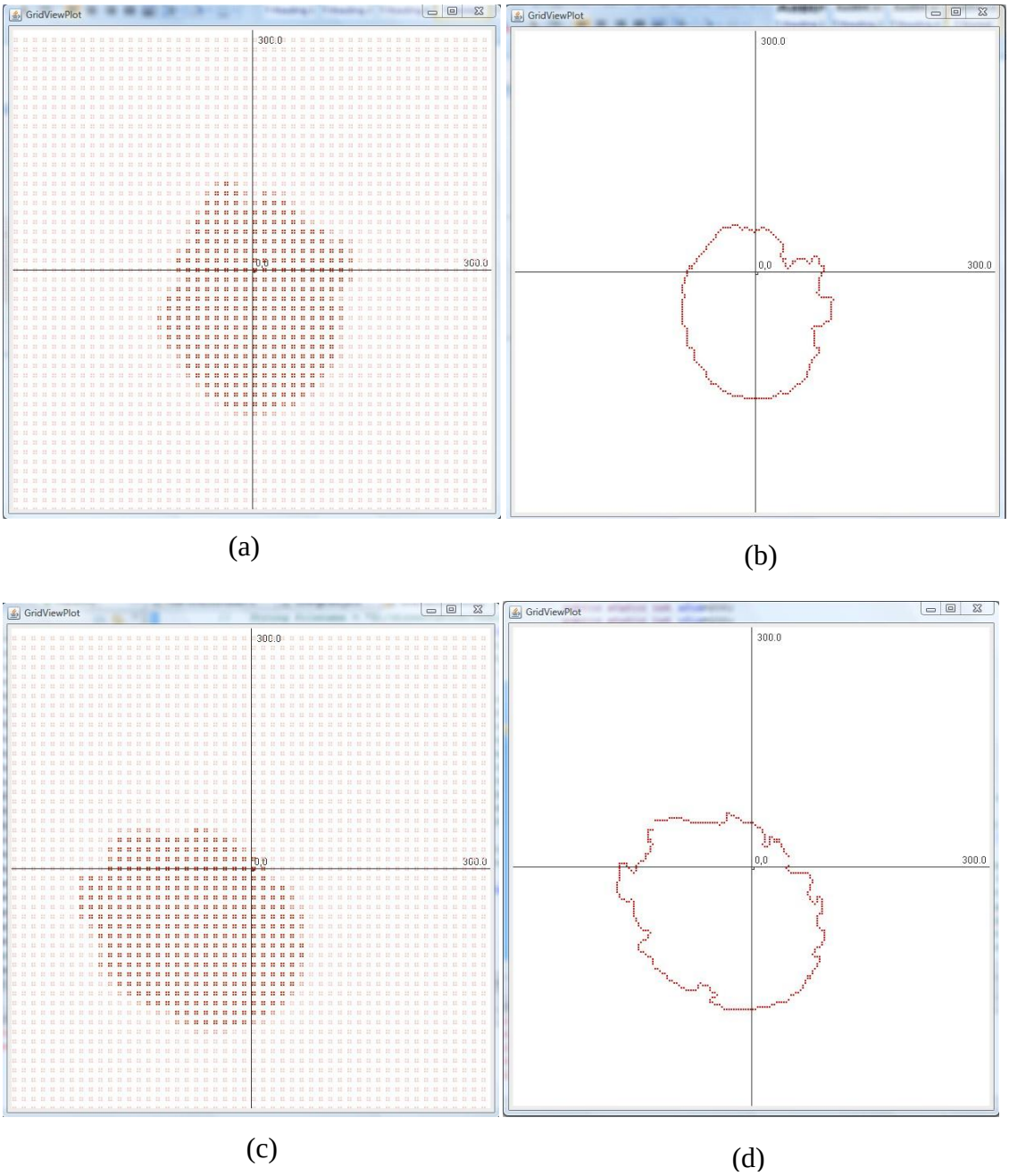


Figure 9.1 Image morphing in DEVS-FIRE. (a) Predicted temperature map; (b) Observation temperature map; (c) Predicted fire front; (d) Warped fire front.

9.3 Improvement of SMC methods of DEVS-FIRE

By importing the image morphing algorithm, we use another way to make use of observation data. Applying it to the SMC methods algorithm, we try to improve the state estimation of DEVS-FIRE. After the step of resampling, we use image morphing

algorithm to warp the selected predicted fire fronts for the use of next step as shown below. From the algorithm above, we can see that, the image morphing also introduces a lot of computations because it will be done for each step. Especially, with increase of the number of line pairs, the computation will be greatly improved. This is the issue of balance between precision and costs. How to balance them to obtain expected results is a problem to be explored.

Algorithm of SMC methods with image morphing

1. Extract corresponding line pairs according to $\{T_i\}_{i=1}^{N}$ and $\{obT_i\}_{i=1}^{N}$
2. For each burning cells C_i in the predicted fire front $fire_t$, find the corresponding position by the all the line pairs, to form the new fire front $fire'_t$

Applying the new algorithm to SMC methods, we intend to improve the simulation results. Therefore, we choose the case 5 in Chapter 7, which has a large difference between the real fire and the filtered fire. After incorporating image morphing, its real fire (display in blue), simulated fire (display in green), and filtered fires (display in red) are displayed in Figure 9.2. By comparing the Figure 9.2 and Figure 7.2(a), we can see the simulation results are greatly improved by applying image morphing.

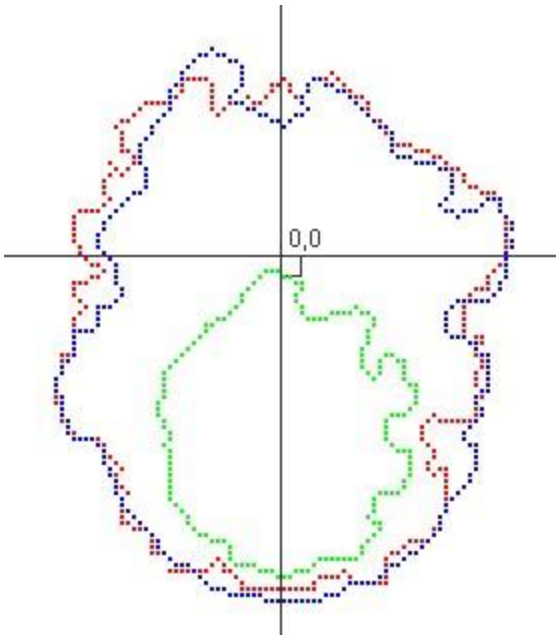


Figure 9.2 Real fire, simulated fire, and filtered fire after using image morphing in SMC methods for case 5 in Chapter 7

CHAPTER 10 VALIDATION OF DEVS-FIRE

Model validation refers to building the right model. There are several methods for model validation but the three general methods are the objective, the subjective, and the combinational method (Sargent 1998). The objective method uses mathematical or statistical tools while the subjective method relies on knowledge of experts or requirements of users. The combinational method is a combination of the objective method and the subjective method. In carrying out model validation, several techniques can be used. These include animation to graphically display the results, comparison with other validated models, and hypothesis test, which utilizes statistics to verify whether the assumptions are correct or not (Sargent 1998).

The developed wildfire spread models, such as, BehavePlus, HFire, and FARSITE, have been validated to some degree using real data (Grabner et al. 1997; Moritz 2003; Arca et al. 2007). Since FARSITE was developed, real data have been collected to test and evaluate using historical wildfire data. In (Finney and Andrews 1994), the horizon prescribed natural fire at Yosemite National Park, which has a well-developed GIS data, was used to initially test the FARSITE simulator. The Howling Fire at Glacier National Park was also used to evaluate the projection method in FARSITE. In (Arca et al. 2007), a fire that occurred in North Sardinia of Italy in the summer 2004 provided another case for validation of FARSITE in a Mediterranean area. BehavePlus, an expansion of Behave fire modeling system, is a wildfire model system to describe fire behaviors, fire effects, and the fire environment. In (Grabner et al. 1997), five models were used to validate the Behave fire behavior in Oak Savannas, and the results show that only fuel model 2 can be used to predict the rate of spread in this area. Another wildfire spread model, HFire

(Highly Optimized Tolerance Fire Spread Model), has been validated for fire spread in California's chaparral (Morais 2001). In (Moritz 2003), the fire history in Santa Monica Mountains verified by the fire patterns, fire return intervals, and fire size distributions provided a test bed for HFire.

One of problems in validating wildfire models is that it is difficult to collect accurate input data from the real system. These data include real-time wind speed, wind direction, fuels, and landscape data. Precisely specifying an area with parameters such as, fuel models, slopes, and aspects, is also difficult since a real geographical space is complicated to describe using a series of numbers. Another issue is collecting accurate real-time wildfire output data such as fire perimeters and areas burned at different time steps. In the validation work of DEVS-FIRE, we use FARSITE, which is a partially validated model (Arca et al. 2007), to validate DEVS-FIRE. This is a step towards the validation of DEVS-FIRE using real data in the future.

10.1 Experimental design

To validate DEVS-FIRE, we used multiple methods to compare its output with that of FARSITE for the same data inputs. We also used 'face validity' in the initial stage of the validation process. This involved a lot of testing to judge that the system behaviors were according to expected fire behaviors. Table 10.1 gives four input sets (corresponding to four experiments) that we use to validate DEVS-FIRE. In the table, WSP and WDR refer to the wind speed and the wind direction respectively. Based on these inputs, we compare two key output parameters, fire spread perimeters and burned areas of DEVS-FIRE at given time steps with those of given by FARSITE. Set 1 was used to test under conditions of uniform fuel, unchanged wind speed and wind direction,

and zero slope landscape. Set 2 has fuel model, unchanged wind speed and wind direction with different aspects and slopes for the landscape. Set 3 has non-uniform cases with three different combinations of fuel models and landscapes, but with unchanged wind speed and wind direction. Set 4 has real GIS landscape data with changing wind conditions. In this case, a wind model is defined to describe the wind speeds and the wind directions at different time steps. Based on these input sets, we compare the outputs of DEVS-FIRE with those of FARSITE, and then draw the conclusions. For the first two input sets, the graphical method, one of the subjective methods, is used to justify DEVS-FIRE as listed in Table 10.2. Because their perimeters increase uniformly, we also used hypothesis test, one of the objective methods to compare the models. For the hypothesis test, we define a statistical variable IPEHH (Increased Perimeter Every Half an Hour), and assume it follows the normal distribution. We utilize χ^2 test, F test, and T test to verify that the IPEHHs of two models follow the same distribution.

Table 10.1 Input sets

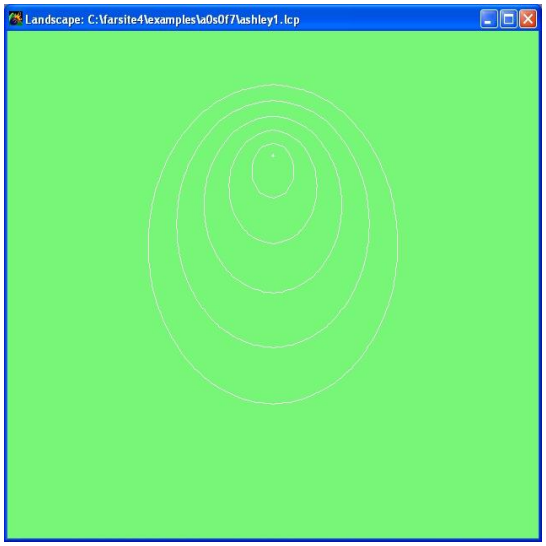
Sets	Slope	Aspect	Fuel	WSP	WDR
Set 1	0	0	7	5	0
	0	0	2	5	0
	0	0	7	5	30
	0	0	7	0-8	0
Set 2	0-25	0	7	5	0
	15	0-360	7	5	0
Set 3	15, 0, 15	180, 0, 0	4, 7, 9	5	0
	15, 0, 15	180, 0, 0	4, 7, 9	5	45
Set 4	Defined	Defined	Defined	Defined	Defined

10.2 The fuels and the wind factors

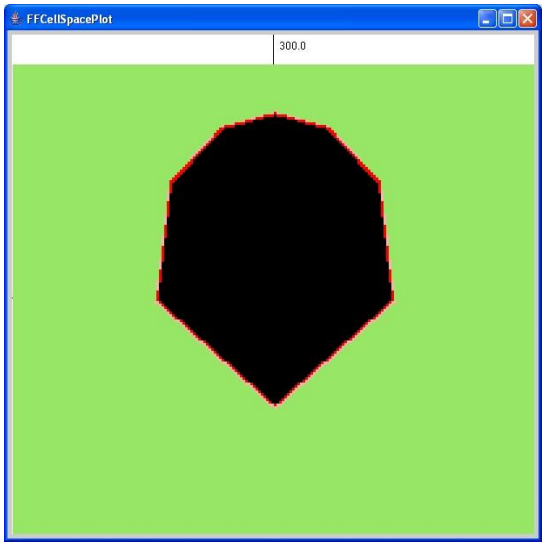
Firstly, we ran simulations in DEVS-FIRE and FARSITE for ten hours based on input set 1 (shown in Table 10.1) to validate the factors of the fuel model and the wind under the condition of zero slope landscape. We first use fuel model 7 and set the wind speed at 5 miles/h, and the wind direction at 0 degree (from north to south), and then change the fuel model from 7 to 2. We also change the wind direction from 0 to 30 degrees. In order to validate against the wind speed factor, we vary the wind speed from 0 to 8 miles/hour. According to these settings, the corresponding results are reported as follows.

Figure 10.1 shows the fire propagation areas of the two models after ten hours. In the figure, the background colors of the window represent the different fuel models. As shown in Figure 10.1(a), the elliptical shapes refer to the fire perimeters every two hours from the beginning in FARSITE. Figure 10.1(b), Figure 10.1(c), and Figure 10.1(d) show the fire perimeters of DEVS-FIRE at the end of the simulation of ten hours of fire spread. From Figure 10.1(a) and Figure 10.1(b), we can see that the fire spread of DEVSFIRE is consistent with that of FARSITE (see also Figure 10.2). However, the head of the fire spread in DEVS-FIRE is in a triangle shape, which is different from the FARSITE's elliptical arc. This is because of the decomposition schema of DEVS-FIRE, while the head of the fire has less decomposition directions as compared to the tail of the fire. Figure 10.1(c) shows the fire perimeter when using a different fuel model (fuel model=2). This results in slow fire spread as compared to the first fuel model (fuel model=7) in Figure 10(b). Figure 10(d) shows the fire perimeter (fuel model=7) when changing the

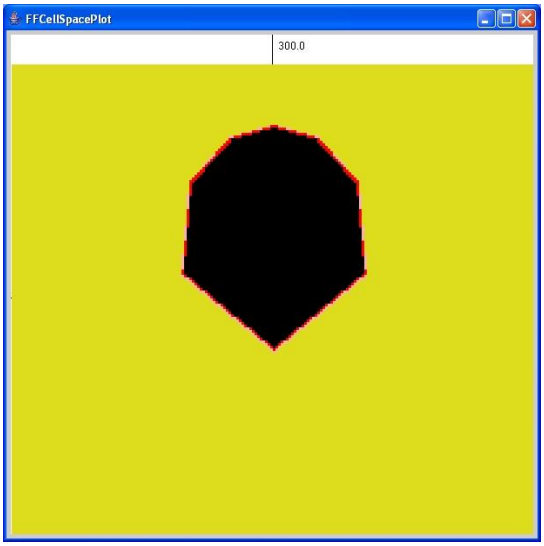
wind direction to 30 degrees. This figure also shows the expected results, and consistent with those of FARSITE.



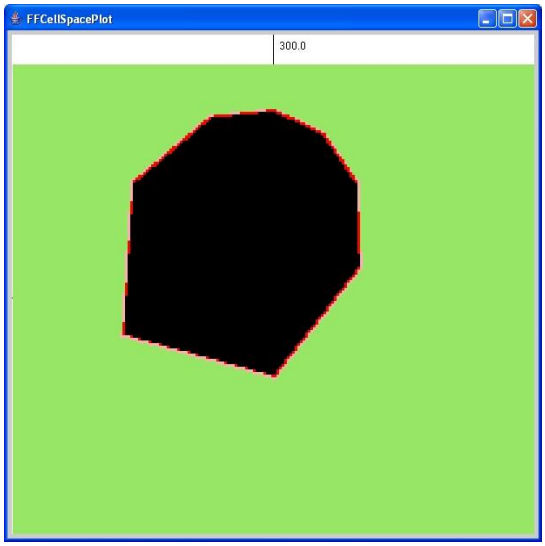
(a) Fuel = 7 of FARSITE



(b) Fuel = 7 of DEVS-FIRE



(c) Fuel = 2 of DEVS-FIRE



(d) WDR = 30 of DEVS-FIRE

Figure 10.1 Uniform cases without aspect and slope

Figure 10.2 plots the time-indexed (every thirty minutes) fire perimeters and areas of DEVS-FIRE and FARSITE for the two fuel models described above. In the figure, DEVSP and FARSITEP refer to perimeters of DEVS-FIRE and FARSITE respectively. Similarly, DEVSA and FARSITEA mean areas of DEVS-FIRE and FARSITE. In DEVS-FIRE, the fire perimeter is calculated based on the sum of outer edges of border cells multiplying the factor of 0.8. This factor of 0.8 is because the sum of the outer edges overestimated the perimeter according to the formula of computing perimeters of an ellipse and its circum-rectangle. Figure 10.3 shows the fire perimeters and areas of ten hours' simulation for fuel model 7 when the wind speed changed from 0 to 8 miles/hour. Results from both DEVS-FIRE and FARSITE are shown and compared. From Figure 3.3, we can see that the perimeters and areas of fire propagation increase with the wind speed. From Figure 10.2 and Figure 10.3, we can see that the fire perimeters and areas of DEVSFIRE show the same trends as those of FARSITE. Specifically, the fire perimeters of DEVS-FIRE are almost the same as those of FARSITE, and the burned areas of DEVS-FIRE may be smaller than those of FARSITE as the fire grows. This is because the two different fire shapes result from the decomposition schema.

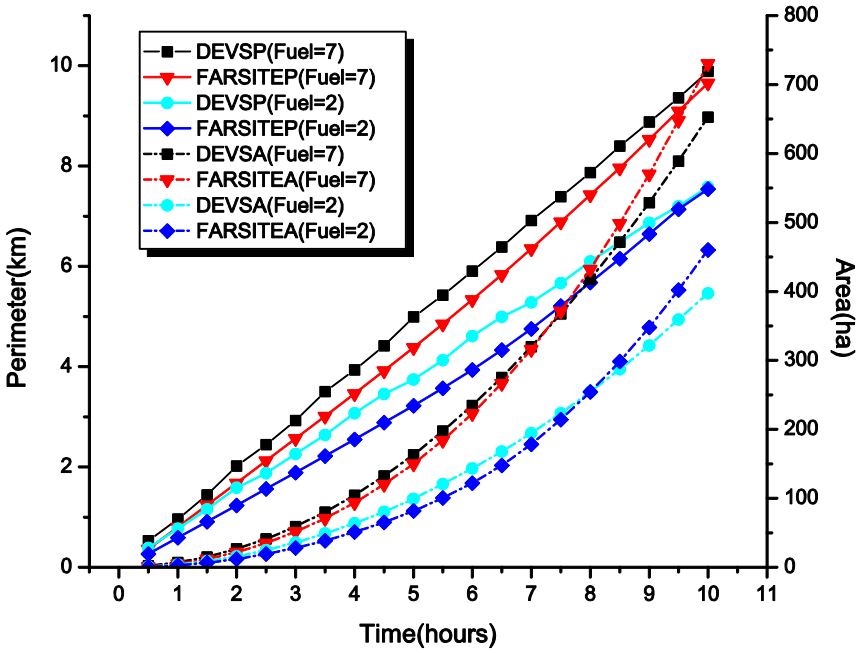


Figure 10.2 Uniform cases without aspect and slope

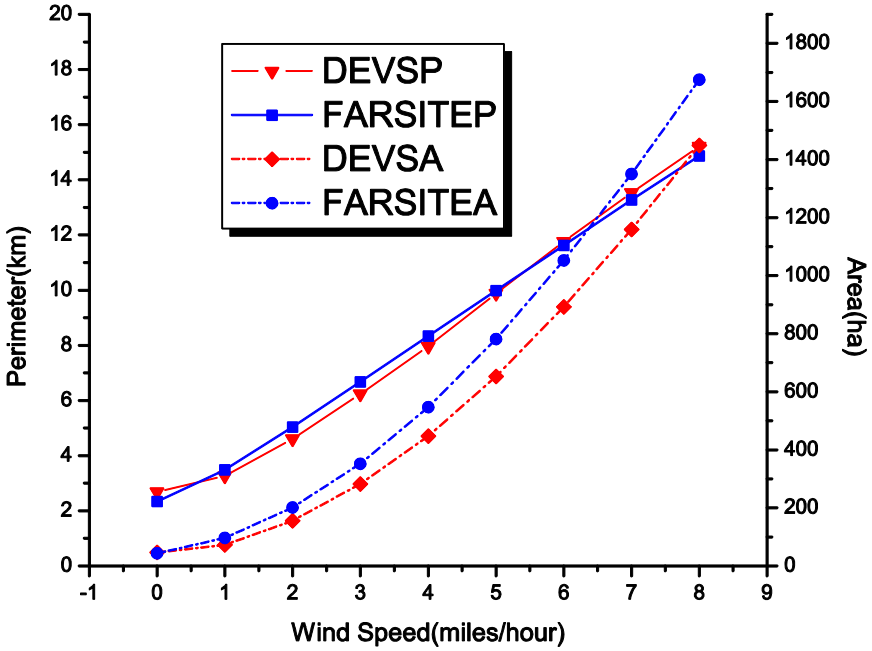
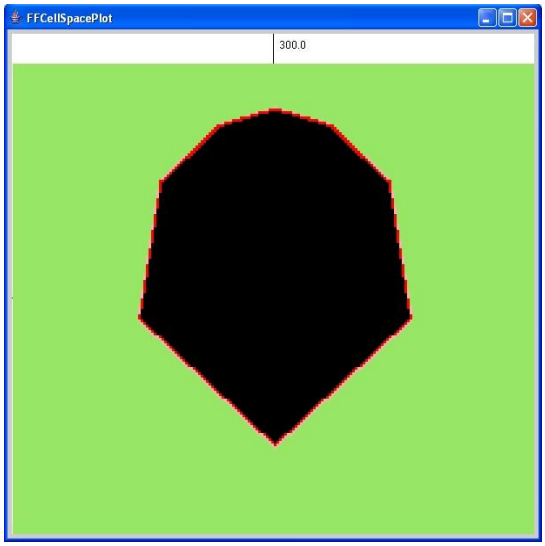


Figure 10.3 Uniform cases with different wind speeds

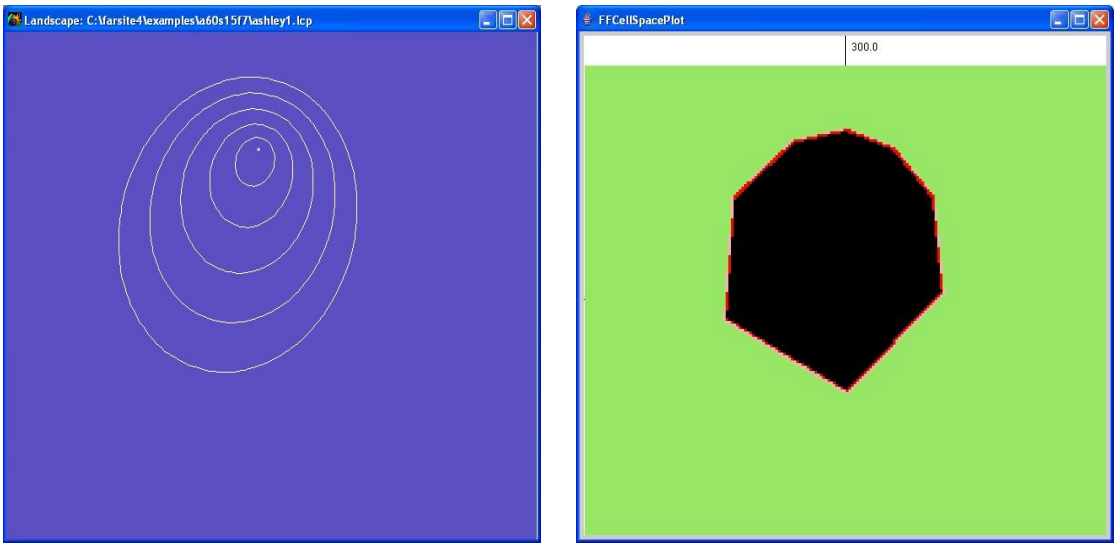
10.3 The slope and the aspect factors

The slope and the aspect (which is defined as the direction the slope is facing-downslope) are two important factors to affect the area and direction of the fire spread. Figure 10.4 indicates the fire propagation in DEVS-FIRE and FARSITE for ten hours with different cases (aspect=0, slope=15; aspect=120, slope=15). From the figures, we can see that the fire spreads slower along southwest when the aspect changes from 0 to 120 degrees. This is because of the downhill slope. According to Figure 10.4, we can conclude that the aspect and the slope influence the direction, increasing or decreasing of the rate of spread (Figure 10.4(b) and Figure 10.1(b)). In both cases, DEVS-FIRE has very similar results to FARSITE. Figure 10.5 and Figure 10.6 show the fire perimeters and areas of DEVS-FIRE and FARSITE at the end of ten hours' simulation when the aspect and slope change (with fuel model 7, wind speed 5 miles/hour, and wind direction 0 degree). In Figure 10.5, the slope is fixed at 15 degrees, and the aspect varies from 0 to 360 degrees. In Figure 10.6, the aspect is 0, and the slope varies from 0 to 25 degrees.

(a) Aspect=0 and Slope=15 of FARSITE



(b) Aspect=0 and Slope=15 of DEVS-FIRE



(c) Aspect=120 and Slope=15 of FARSITE (d) Aspect=120 and Slope=15 of DEVS-FIRE

Figure 10.4 Uniform cases with aspect and slope

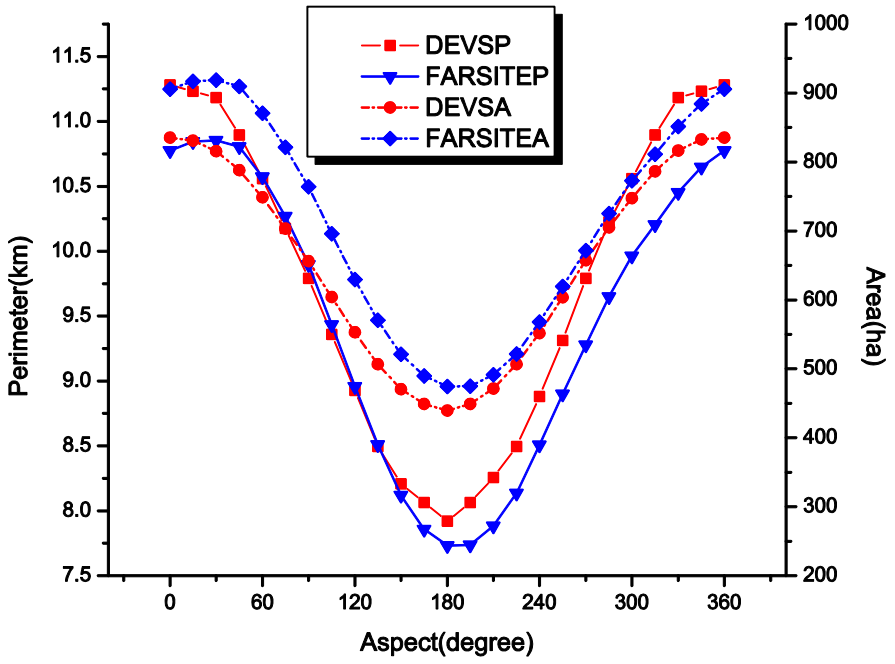


Figure 10.5 Data with different aspects

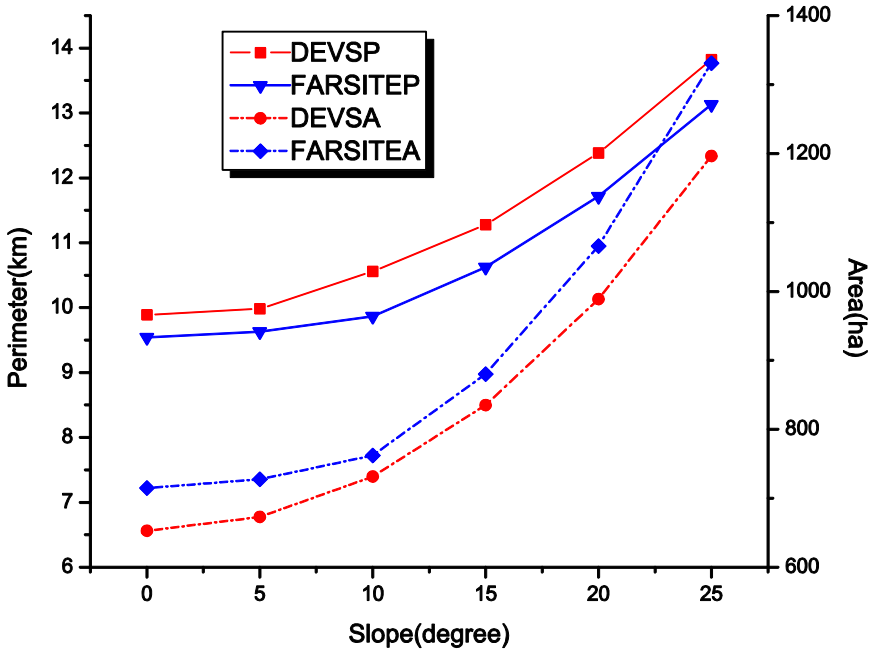


Figure 10.6 Data with different slopes

From Figure 10.6, we observe that when the wind blows uphill, the larger the slope is, the faster the fire spreads. This is expected. Figure 10.5 shows how the aspects change the fire spread direction and rate of spread. Specifically, the rate of spread is the fastest when the wind blows uphill (aspect=0, 360 degrees in Figure 10.5). The rate of spread is the slowest when the wind blows downhill (aspect=180 in Figure 10.5). The fire rate of spread is symmetric with respect to the aspect of 180 degrees. Table 10.2 shows the results of hypothesis tests for input set 1 and set 2. In the table, 1~5 stand for five different input combinations used in Table 10.2; GF is the abbreviation of Goodness of Fit test; F and T refer to the values of F test and T test. According to Table 10.3, the values of χ^2 for all the cases of DEVS-FIRE and FARSITE are less than the critical value of χ^2 (1, 0.975)=5.02, which means the IPEHH of every case follows the normal distribution with the significance level 0.025. Because $F_{0.975}(19, 19) = 0.29$, $F_{0.025}$ (19, 19)

= 3.43, all *F* values in Table 10.2 fall in this critical region. Therefore, they accept the assumptions $\sigma_1^2=\sigma_2^2$. Finally, according to Table 10.3, all the absolute values of *t* are larger than $t_{0.975}$ (38) = 2.024, and it can be said that the means of the IPEHH are the same at the significance level 0.05. Therefore, we can conclude that DEVS-FIRE has the same perimeters as those of FARSITE under the same conditions for uniform cases.

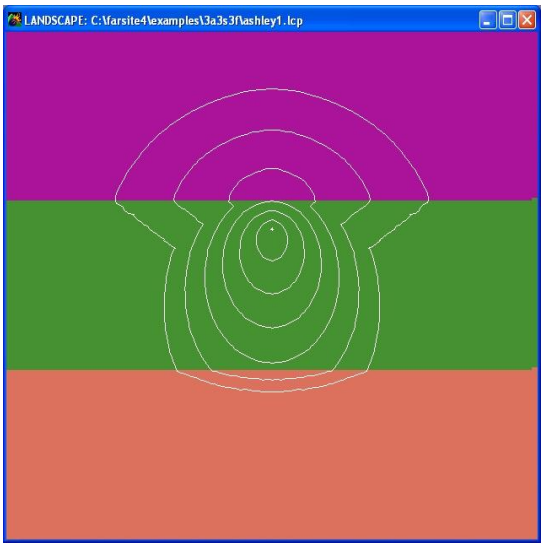
Table 10.2 Hypothesis test results

Test	1	1	3	4	5
GF DEVS-FIRE	0.08	0.13	0.06	0.09	0.12
GF FARSITE	0.12	0.27	0.12	0.14	0.12
F	0.72	0.51	0.53	0.70	1.05
T	0.74	0.11	-0.73	0.93	1.38

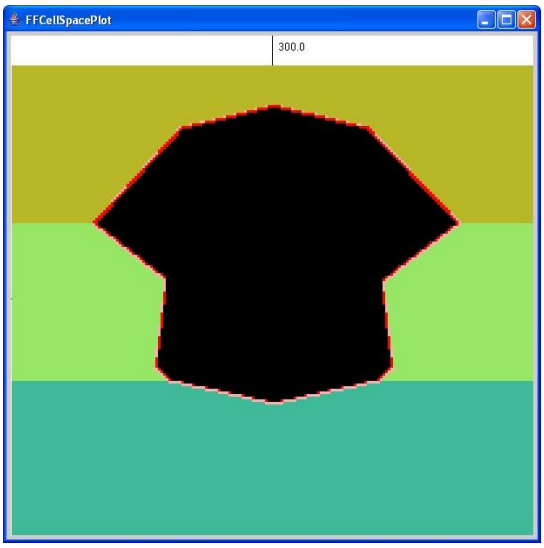
10.4 Non-uniform fuel/slope/aspect

In this experiment, we define an area with three zones as shown in Figure 10.7(a). The top zone has fuel model of 4 with slope 15 degrees and aspect 180 degrees; the middle zone has fuel model of 7 with slope 0 degree and aspect 0 degree; the bottom zone has fuel model of 9 with slope 15 degrees and aspect 0 degree. Two simulations were run. The first one has wind speed 5 miles/hour and wind direction 0 degree. The second one has wind speed 5 miles/hour and wind direction 45 degrees. Figure 10.7(a) and 10.7(b) show the results of FARSITE and DEVS-FIRE respectively of ten hours' fire spread in the first simulation. Figure 10.7(c) and 10.7(d) show the corresponding results of the second simulation. Table 10.3 lists the data of fire perimeters and burned areas hourly from 5 to 9 hours. In the table, A, D, F, P, and WDR refer to area (ha), DEVS-FIRE, FARSITE, perimeter (km), and wind direction (degrees) respectively. From the

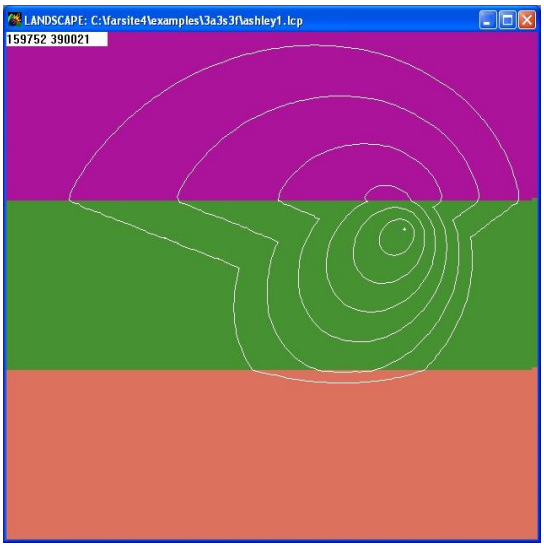
figures, we can see that the fire shapes of DEVS-FIRE and FARSITE are very much similar. The data in the table also confirm that the two simulation models give the consistent results.



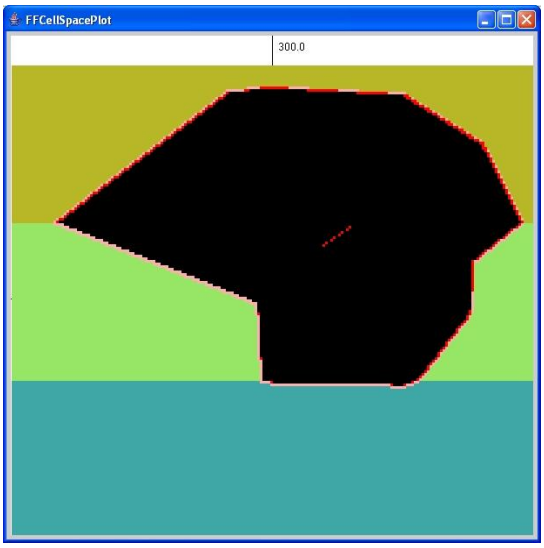
(a) WDR = 0 of FARSITE



(b) WDR = 0 of DEVS-FIRE



(c) WDR = 450 of FARSITE



(d) WDR = 0 of DEVS-FIRE

Figure 10.7 Non-uniform cases

Table 10.3 GIS data with non-uniform fuel/slope/aspect

WDR			5	6	7	8	9
0	P	F	4.4	5.8	7.5	9.3	11.2
		D	5.4	6.8	8.2	9.6	11.1
	A	F	147.9	238.1	373.7	555.6	788.7
		D	174.8	278.2	408.9	561.3	739.5
45	P	F	4.9	6.8	9.1	11.5	14.0
		D	5.6	7.9	10.1	12.2	14.2
	A	F	164.9	296.3	507.8	804.9	1174.2
		D	229.1	387.5	599.4	864.1	1159.7

10.5 GIS data and varying wind condition

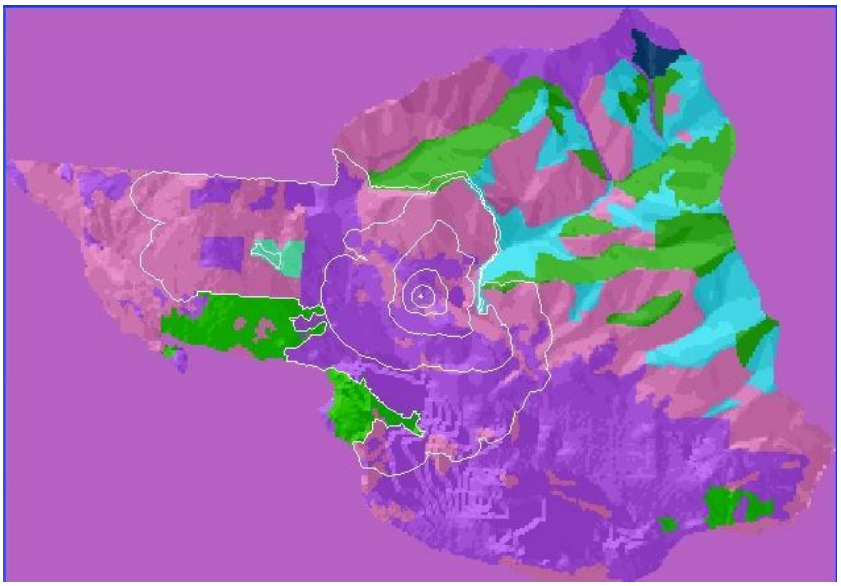
This is the case that is close to reality where fuel model, aspect, and slope vary from cell to cell, and wind conditions change in a timely manner. This experiment uses the GIS data from the Ashley project (an example provided by FARSITE 4.0), and we ran the simulations for ten hours. In the experiment, random wind speeds (around 5 miles/hour) and wind directions are generated every one-hour for the first eight hours. For testing purpose, we also manually increase the wind speed to around 16 miles/hour in the last two hours. Figure 10.8(a) and 10.8(b) show the simulation results of FARSITE and DEVS-FIRE respectively.

Table 10.4 shows the perimeters and areas of FARSITE and DEVS-FIRE respectively. According to the figures, the fire fronts spread from the center of the space to northwest, and DEVS-FIRE and FARSITE have the similar spread patterns. As can be

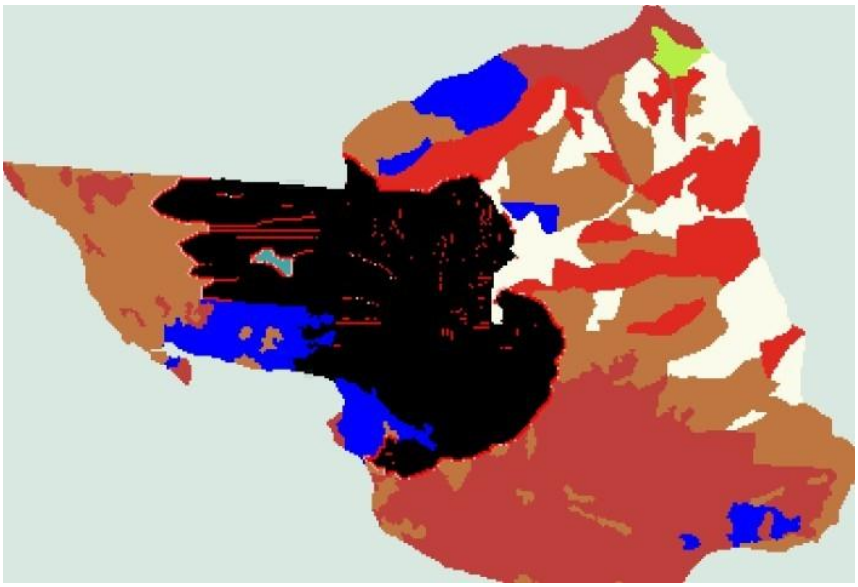
seen from the table, the outputs greatly increase from the ninth hour to the tenth hour because the wind speed changed from 5 to 18 miles/hour. From both the figure and the table, we can draw the conclusion that DEVS-FIRE has the similar simulation results as FARSITE for GIS data.

Table 10.4 GIS data with wind model

Time		6	7	8	9	10
P	F	4.9	7.1	10.5	13.7	27.5
	D	5.9	7.9	10.6	14.0	26.6
A	F	136.8	251.5	486.6	695.7	1385.3
	D	201.7	308.9	542.3	714.8	1239.6



(a) FARSITE



(b) DEVS-FIRE

Figure 10.8 GIS data with wind model

In the chapter, we validate DEVS-FIRE, by comparing its results with those of FARSITE under the same input conditions. Both graphical and statistical approaches are used to compare the fire perimeters and burned areas in DEVS-FIRE and FARSITE. The comparisons show that simulation results from DEVS-FIRE are consistent with those of FARSITE, including fire spread direction. The work builds a solid ground for validation of DEVS-FIRE using real historical fire data. Through this work, the DEVS-FIRE model will be improved by adjusting the model coefficients, exploring the precise wind adjustment factors for different fuel models, and optimizing the fire spread decomposition schema.

CHAPTER 11 DISCUSSIONS AND FUTURE WORK

11.1 Discussions

Assimilating data into wildfire simulation is a challenging task due to the non-linear, non-Gaussian wildfire behavior and the large-scale spatial temporal system state of wildfires. We apply SMC methods to data assimilation for wildfire spread simulation by estimating the evolving fire front from ground temperature sensor data. As an important step, this work focuses on developing the basic system components and their models and algorithms that work together for applying SMC methods to data assimilation in discrete event wildfire simulation. This work builds a foundation where more advanced SMC methods-based algorithms (such as different sampling, resampling, and weight computation algorithms) and more in-depth analysis can be incorporated in the future. Next we discuss several important aspects of this work as well as some directions for future work.

The results of SMC methods-based wildfire data assimilation depend on many different factors. Among them the number of particles can significantly affect the quality of data assimilation. In this work, each particle represents a fire front shape, from which the fire front in the next step is generated. Thus the larger the number of particles, the more possible fire shapes can be generated. This increases the “diversity” of the fire fronts. In this work, we used 200 particles. We think this is an accepted number because the fire shapes are relatively small due to the relatively short simulation duration (6 hours of simulation). However, to assimilate data in a longer simulation, the fire shape will be significantly larger. We expect a larger number of particles are needed in that case in order to obtain quality results.

Noise generation results in uncertainty and diversity for possible fire fronts in the sampling stage of SMC methods. Two approaches can be considered for generating noises in DEVS-FIRE, including incorporating noises to the parameters during the computation of Rothermel's model (such as, rate of speed (*ros*), effective wind speed (*efw*), and spread direction (*sdr*)), and directly adding graph noises to the fire front, to generate a new one as illustrated in our algorithm. Although, we divide the fire front into multiple segments to generate noises, compared with the numeric variable, the graph noise is much more complicated and hard to introduce enough "diversity".

Weight computation in the resampling stage is used to decide the optimal fire fronts by comparing the generated fire fronts with the measurements, temperatures of all the deployed sensors. Therefore, the correctness of the measurement model is important, converting the fire front to temperatures of all the sensors in real applications. Because the information of all the sensors decides the weight of a fire front, reasonably calculating the contribution of each sensor to the total effect is needed. We introduced side effects (probability less than the constant *a*) for weight computation. All the temperatures of the deployed sensors form the temperature map of the space. To effectively decide the effect of sensors, we can assign different sensitivities to them by differentiating the head and the tail of the fire front because the head of the fire is more important to study fire propagation.

The performance issue needs to be addressed because of the SMC methods algorithm and the simulation model of DEVS-FIRE. On the one hand, DEVS-FIRE itself consumes a lot of computation resources and memory at runtime due to the number of atomic models and their communications. On the other hand, SMC methods algorithm

greatly introduces computations by generating particles and running multiple simulation models. Specifically, if we run simulations with long duration and a big number of particles, the performance issue is more challenging. By reducing the resolution of the cell space, we can achieve speedups for execution of single simulation of DEVS-FIRE. Parallel and distributed computation could be utilized to solve the problem of multiple particles by assigning tasks to different computers or processors.

Although the simulation results are improved by applying image morphing into SMC methods, it is still necessary to explore more to further improve the results. This could be achieved by extracting more feature pair lines, for example, divide the temperature map into more segments, and extract one line from each segment. Especially, for the fire head, it should be further studied.

11.2 Conclusions and future work

In this work, a framework of dynamic data driven application system for wildfire spread simulation is designed. Basically, the physical model wildfire spread of DEVS-FIRE is used to predict the fire growth if the slope, aspect, fuel, weather of a fire field are known. Although it has been partially validated using another model of FARSITE, it still has some drawbacks to precisely capture the real situation of the fire propagation. Therefore, the framework of dynamic data driven is introduced so that the real time data is incorporated into the physical model. The measurement model is used to convert the outputs of DEVS-FIRE into the data, which can be compared with real time data, thus to evaluate and choose optimal predictions. Sequential Monte Carlo methods, the data assimilation approach used in the work, are an analysis technique using a series of statistical methods in dynamic systems to recursively estimate the probability density

function, from which the system states can be computed and estimated. Therefore, the non-linear state space model is formalized based on DEVS-FIRE and the measurement model, and solved by sequential Monte Carlo methods.

To justify the system, identical twin systems are designed and a couple of experiments results are obtained. From the experimental results, we know that by applying sequential Monte Carlo methods to the DEVS-FIRE, the simulation results could be improved. To examine influences of different factors on the simulation results, several sets of experiments are executed. Also, we used a series of tools to study and analyze the procedure of SMC methods, thus to provide guideline for improving application of SMC methods in DEVS-FIRE. Finally, the image morphing algorithm was introduced to improve the simulation results for the worse case of DDDAS in DEVS-FIRE.

The future work should be improving the algorithm to make the SMC methods algorithm to converge to one or more big weights during the state estimation. Firstly, we can improve the performance so that it can support more particles so that more possibilities can be generated to gain “diversity”. Also, the value of noises affects the generated fire fronts. To empirically confirm the noise range seems another direction for “diversity”.

REFERENCES

Allen, G. 2007. Building a dynamic data driven application system for hurricane forecasting. International Conference on Computational Science (1) 2007:1034-1041.

Anderson J.L. 2001. An ensemble adjustment filter for data assimilation. *Monthly Weather Rev*, 129:2884-903.

Andrews, P.L., Bevins, C.D., and Seli, R.C. 2005. BehavePlus fire modeling system, version 3.0: User's Guide Gen. Tech. Rep. RMRS-GTR-106WWW Revised. Ogden, UT: Department of Agriculture, Forest Service, Rocky Mountain Research Station, 132.

Antoniou, C., Ben-Akiva, M., and Koutsopoulos, H.N. 2007. Nonlinear Kalman filtering algorithms for on-line calibration of dynamic traffic assignment models. *Intelligent Transportation Systems, IEEE Transactions*, 4:661-670.

Arca B., Duce, P., Pellizzaro, G., Laconi, M., Salis, M., and Spano, D. 2007. Evaluation of Farsite simulator in a Mediterranean area. The 4th International Wildland Fire Conference, Seville, Spain.

Azzabou, N., Paragios, N., and Guichard, F. 2005. Application of particle filtering to image enhancement. CERTIS 05-18.

Bei, N., de Foy, B., Lei, W., Zavala, M., and Molina, L.T. 2008. Using 3DVAR data assimilation system to improve ozone simulations in the Mexico City Basin. *Atmos. Chem. Phys.*, 8:7353-7366.

Beier, T., and Neely, S. 1992. Feature-based image metamorphosis. Proceedings of SIGGRAPH'92, *Comput. Graph*, 26:35-42.

Borga, M. 2002. Accuracy of radar rainfall estimates for streamflow simulation. *J. Hydrol.*, 267:26-39.

Bouttier, F., and Courtier, P. 1999. Data assimilation concepts and methods. Training course notes of ECMWF.

Bradley, J.M. 2007. Particle filter based mosaicking for forest fire tracking. Master thesis, Brigham Young University.

Chen, T., Morris, J., and Martin, E. 2008. Dynamic data rectification using particle filters. *Computers and Chemical Engineering*, 32:451-462.

Chen, R. 2004. Sequential Monte Carlo methods and their applications: An overview and recent developments. Representation to Institute for Mathematical Sciences National University of Singapore.

Coen, J.L., Beezley, J.D., Bennethum, L.S., Douglas, C.C., Kim, M., Kremens, R., Mandel, J., Qin, G., and Vodacek, A. 2007. A wildland fire dynamic data-driven application system. 11th Symposium on Integrated Observing and Assimilation Systems for the Atmosphere, Oceans, and Land Surface (IOAS-AOLS).

Crisan, D. 2001. Particle filters—A theoretical perspective. *Sequential Monte Carlo Methods in Practice* (eds A. Doucet, J. F. G. de Freitas and N. J. Gordon). New York: Springer-Verlag.

Crisan, D., P. Del Moral, and T. Lyons. 1999. Discrete filtering using branching and interacting particle systems. *Markov Proc. Rel. Fields*, 5:293-318.

Crisan, D., and A. Doucet. 2002. A survey of convergence results on particle filtering methods for practitioners. *IEEE Transactions on Signal Processing*, 50(3):736-746.

Darema, F. 2000. Dynamic data driven application systems (Symbiotic measurement & simulation systems): A new paradigm for application simulations and a new paradigm for measurement systems. NSF sponsored workshop.

Darema, F. 2007. Introduction to the ICCS 2007 workshop on dynamic data driven applications systems. International Conference on Computational Science (1) 2007: 955-962.

Doucet, A., Gordon, N.J., and Krishnamurthy, V. 1999. Particle filters for state estimation of jump Markov linear systems, Technical Report CUED/FINFENG/TR 359, Cambridge University Engineering Department.

Douglas, C.C., Beezley, J.D., Coen, J., Li, D., Li, W., Mandel, A.K., Mandel, J., Qin, G., and Vodacek, A. 2006. Demonstrating the validity of a wildfire DDDAS. Computational Science - ICCS 2006, Alexandrov, V. N., van Albada, G. D., Sloot, P. M. A., and Dongarra, J. J. (eds.), Springer-Verlag, Berlin, 3:522-529.

Douglas, C.C., Lodder, R.A., Ewing, R.E., Efendiev, Y., Qin, G., Coen, J., Kritz, M., Beezley, J.D., Mandel, J., Iskandarani, M., Vodacek, A., and Haase, G. 2006. DDDAS approaches to wildland fire, modeling and contaminant tracking. Proceedings of the 2006 Winter Simulation Conference, 2117-2124.

El-Osery, A., Burge, J., Jamshidi, M., Fathi, M., and Akbarzadeh-T, M. -R, 2002. V-LAB - A virtual laboratory for autonomous agents - SLA based Controllers. *IEEE Transaction on Systems, Man and Cybernetics*, 32(6):791-803.

Farhat, C., Michopoulos, J., Chang, F.K., Guibas, L.J., and Lew, A.J. 2006. Towards a dynamic data driven system for structural and material health monitoring. International Conference on Computational Science (3) 2006:456-464.

Finney, M.A., and Andrews, P.L. 1994. The FARSITE fire area simulator: fire management applications and lessons of summer 1994. Proceedings of the 1994 Interior West Fire Council Meeting and Program, 209-216.

Finney, M.A. 1998. FARSITE: fire area simulator–model development and evaluation. United States Department of Agriculture Forest Service Rocky Mountain Research Station Research Paper, RMRS-RP-4 Revised March 1998, revised February 2004.

Fox, D., Thrun, S., Burgard, W., and Dellaert, F. 2001. Particle filters for mobile robot localization. *Sequential Monte Carlo Methods in Practice* (eds A. Doucet, J. F. G. de Freitas and N. J. Gordon). New York: Springer-Verlag.

Fujimoto, R.M., Guensler, R., Hunter, M., Kim, H.K., Lee, J., Leonard, J., Palekar, M., Schwan, K., and Seshasayee, B. 2006. Dynamic data driven application simulation of surface transportation systems. International Conference on Computational Science (3) 2006:425-432.

Glinsky, E., and Wainer, G. 2004. Modeling and simulation of hardware/software systems with CD++. Proceedings of the 2004 Winter Simulation Conference.

Gordon, N.J., Salmond, D.J., and Smith, A.F.M., 1993. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proceedings on Radar and Signal Processing*, 140:107–113.

Grabner, K.G., Dwyer, J.P., and Cutter, B.E. 1997. Validation of BEHAVE fire behavior predictions in Oak Savannas using five fuel models. Proc. Eleventh Central Hardwood Conf., Univ. of Missouri, Columbia, 202-215.

Grossi, P. 2007. The 2007 U.S. wildfire season lessons from southern California. Last accessed in Aug. 2009 from http:// www.rms.com/Publications/2007_US_Wildfire_Season.pdf.

Gu, F., and Hu, X. 2008. Towards applications of particle filters in wildfire spread simulation. In: Proceedings of the 40th Conference on Winter Simulation, 2852-2860.

Gu, F., Yan, X., and Hu, X. 2009. State estimation using particle filters in wildfire spread simulation. 42nd Annual Simulation Symposium (ANSS'09).

Gustafsson, F., Gunnarsson, F., Bergman, N., Forssell, U., Jansson, J., Karlsson, R., and Nordlund, R.-J. 2002. Particle filters for positioning, navigation and tracking. *IEEE Transactions on Signal Processing*, 50(2): 425–437.

Hu, X., Sun, Y., and Ntaimo, L. 2010. DEVS-FIRE: Design and formal specification of discrete event wildfire spread and suppression models. *SIMULATION*, submitted.

Huang, D., Sarjoughian, H.S., Wang, W., Godding, G., Rivera, D.E., Kempf, K.G., and Mittelmann, H. 2009. Simulation of semiconductor manufacturing supply-chain systems With DEVS, MPC, and KIB. *IEEE Transactions on Semiconductor Manufacturing*, 22(1):164-174.

Jazwinski, A.H. 1970. *Stochastic processes and filtering theory*. Mathematics in science and engineering. Academic Press, New York, USA.

Kailath, T., Sayed, A.H., and Hassibi, B. 2000. *Linear estimation. Information and System Sciences Series*. Prentice Hall, Upper Saddle River, NJ, USA.

Keeley, J.E. 1995. Future of California floristics and systematics: Wildfire threats to the California Flora. *Madrono*, 42:175-179.

Kennard, D. 2008. Present-Day wildfires, accessed in Oct. 1009 from http://www.forestencyclopedia.net/p/p1470.

Kremens, R., Faulring, J., Gallagher, A., Seema, A., and Vodacek, A. 2003. Autonomous field-deployable wildland fire sensors. *International Journal of Wildfire*, 12.

Le Goc, M., and Frydman, C. 2003. The Discrete Event Concept as a Paradigm for the "Perception Based Diagnosis" of Sachem. *Journal of Intelligent & Robotic Systems, Theory and Applications*, 4 (2):207-232.

Liu, J.S. 1996. Nonparametric hierarchical Bayes via sequential imputations. *The Annals of Statistics*, 24:911-930.

Lindsey, R. 2008. Amazon fires on the rise. Earth Observatory (NASA).

Linn, R., Reisner, J., Colman, J.J., and Winterkamp, J. 2002. Studying wildfire behavior using FIRETEC. *Int. J. of Wildland Fire*, 11:233-246.

Mandel, J., Chen, M., Franca, L.P., Johns, C., Puhalskii, A., Coen, J.L., Douglas, C.C., Krements, R., Vodacek, A., and Zhao, W. 2004. A Note on Dynamic Data Driven Wildfire Modeling. *Computational Science-ICCS 2004* (eds M. Bubak, G. D. van Albada, P. M. A. Sloot, and J. J. Dongarra). Berlin: Springer.

Mandel, J., Bennethum, L.S., Beezley, J.D., Coen, J.L., Douglas, C.C., Kim, M., and Vodacek, A. 2008. A wildland fire model with data assimilation. *Math. Comput. Simul.*, 79(3).

Mandel, J., Beezley, J.D., Coen, J.L., and Kim, M. 2009. Data assimilation for wildland fires: Ensemble Kalman filters in coupled atmosphere-surface models. *IEEE Control Magazine Systems* 3, 29:47-65.

Mihaylova, L., Angelova, D., Honary, S., Bull, D., Canagarajah, N., and Ristic, B., 2007. Mobility tracking in cellular networks using particle filtering. *IEEE Trans. Wireless Communications*, 6(10).

Morais, M. 2001. Comparing spatially explicit models of fire spread through Chaparral fuels: A new model based upon the Rothermel fire spread equation. MA Thesis, University of California, Santa Barbara.

Moritz, M.A. 2003. Lessons from the October 2003 wildfires in southern California. Last accessed in Oct. 2007 from http://www.physics.ucsb.edu/˜complex/max/hfiresamo.html#output.

Ntaimo, L., Hu, X., and Sun., Y. 2008. DEVS-FIRE: Towards an integrated simulation environment for surface wildfire spread and containment. *SIMULATION: Transactions of The Society for Modeling and Simulation International*, 84(4):137-155.

Oden, J.T., Diller, K.R., Bajaj, C.L., Browne, J.C., Hazle, J., Babuska, I., Bass, J., Demkowicz, L.F., Feng, Y., Fuentes, D., Prudhomme, S., Rylander, M.N., Stafford, R.J., and Zhang. Y. 2006. Development of a computational paradigm for laser treatment of cancer. International Conference on Computational Science (3) 2006:530-537.

Palaniappan, S., Sawhney, A., and Sarjoughian, H.S. 2006. Application of the DEVS framework in construction simulation. Proceedings of the 2006 Winter Simulation Conference.

Pastor, E., Zarate, L., Planas, E., and Arnaldos, J. 2003. Mathematical models and calculations systems for the study of wildland fire behavior. *Prog. Energy. Combust. Sci.*, 29:139-153.

Qiao, G., Riddick, F., and McLean, C. 2003. Data driven design and simulation system based on XML. Proceedings of the 2003 Winter Simulation Conference, eds. S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, 1143-1148. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Rey, M. 2004. USDA Testimony to Congress. Last accessed in Aug. 2009 from http://www.fs.fed.us/congress/108/house/oversight/rey/050504.html.

Rothermel, R.C. 1972. A mathematical model for predicting fire spread in wildland fuels. USDA For. Serv. Res. Pap. INT-115.

Running, S.W. 2008. Ecosystem disturbance, carbon, and climate. *Science*, 321:652-653.

Sargent, R.G. 1998. Verification and validation of simulation models. Proceedings of The 1998 Winter Simulation Conference, 121-130.

Smythe, D.B. 1990. A two-pass mesh warping algorithm for object transformation and image interpolation. Technical Report 1030, ILM Computer Graphics Department, Lucasfilm, San Rafael, Calif.

Tannock, J., Cao, B., Farr, R., and Byrne, M. 2007. Data-driven simulation of the supply-chain--Insights from the aerospace sector. *International Journal of Production Economics*, 110 (1-2):70-84.

Uhrmacher, A.M., and Priami, C.,2005. Discrete event systems specification systems biology - a discussion of stochastic PI calculus and DEVS. Proceedings of the 2005 Winter Simulation Conference.

Van Wagner, C.E. 1973. Height of crown scorch in forest fires. *Canadian Journal of Forest Research*, 3(3):373-378.

Van Wagner, C.E. 1975. Convection temperatures above low intensity forest fires. *Res. Can For. Serv.*, 31(2):21.

Wang, W. 2004. Data-driven techniques for hardware and software synthesis for embedded systems. Ph.D. Dissertation, Princeton University, USA.

Weber, RO. 1991. Modeling fire spread through fuel beds. *Progress Energy Combustion Science*, 17:67-82.

Wilkin, J., Zavala-Garay, J., Levin, J., and Zhang, W.G. 2008. Four-dimensional variational assimilation of satellite temperature and sea level data in the coastal ocean and adjacent deep sea. IGARSS 2008.

Wolberg, G. 1990. *Digital image warping*. IEEE Computer Society Press, Los Alamitos, Calif.

Yan, X., Gu, F., Hu, X., and Guo, S. 2009. A dynamic data driven application system for wildfire spread simulation. Proceedings of Winter Simulation Conference 2009.

Zeigler, B.P., Moon, Y., Kim, D., and Kim, J.G. 1996. DEVS/C++ A High Performance Modelling and Simulation Environment. 29th Annual Hawaii International Conference on System Science.

Zeigler, B.P., Praehofer, H., and Kim, T.G. 2000. *Theory of modeling and simulation, 2nd Edition*, Academic Press.

Zeigler, B.P., Sarjoughian, H. 2002. Introduction to DEVS modeling and simulation with JAVA: A simplified approach to HLA-compliant distributed simulations. The University of Arizona, Tucson, Arizona, USA.

Zhang, J., Chen, R., Tang, C., and Liang, J. 2003. Origin of scaling behavior of protein packing density: A sequential Monte Carlo study of compact long chain polymers. *Journal of Chemical Physics*, 118(13): 5102-610.